REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1999	MORS Paper

4. TITLE AND SUBTITLE
DYNAMIC ROUTING OF UNMANNED AERIAL VEHICLES USING REACTIVE TABU SEARCH

5. FUNDING NUMBERS

6. AUTHOR(S)
Kevin P. O'Rourke, T. Glenn Bailey, Raymond Hill, William B. Carlton

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology
Department of Operational Sciences
2950 P Street
Wright Patterson AFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GOA/ENS/99M-06

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Lt Col Mark A. O'Hair, Chief, Systems Integration Division (DOM)
Unmanned Aerial Vehicle Battlelab
1003 Nomad Way Ste 107
Eglin AFB FL 32542-6867

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

20000804 183

13. ABSTRACT *(Maximum 200 words)*

In this paper we consider the dynamic routing of unmanned aerial vehicles (UAVs) currently in operational use with the US Air Force. Dynamic vehicle routing problems (VRP) have always been challenging, and the airborne version of the VRP adds dimensions and difficulties not present in typical ground-based applications. Previous UAV routing work has focused on primarily on static, pre-planned situations; however, scheduling military operations, which are often ad-hoc, drives the need for a dynamic route solver that can respond to rapidly evolving problem constraints. With these considerations in mind, we examine the use of a Java-encoded metaheuristic to solve these dynamic routing problems, explore its operation with several general problem classes, and look at the advantages it provides in sample UAV routing problems. The end routine provides routing information for a UAV virtual battlespace simulation and allows dynamic routing of operational missions.

14. SUBJECT TERMS
Air Force Research, Operations Research, Optimization, Combinatorial Analysis, Algorithms, Remotely Piloted Vehicles, Surveillance Drones, Tabu Search, Vehicle Routing Problem, Java, Heuristics, Traveling Salesman Problem

15. NUMBER OF PAGES
40

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Dynamic Routing of Unmanned Aerial Vehicles Using Reactive Tabu Search

Kevin P. O'Rourke
T. Glenn Bailey
Raymond Hill
William B. Carlton

Air Force Institute of Technology
Department of Operational Sciences
2950 P St., Bldg 640
Wright-Patterson AFB OH, 45433-7765
Voice 937.255.6565 x4332
Fax 937.656.4943
Email Ray.Hill@afit.af.mil

67th MORS Symposium
Working Group 10 -- Unmanned Systems
26 November 1999

ABSTRACT

In this paper we consider the dynamic routing of unmanned aerial vehicles (UAVs) currently in operational use with the US Air Force. Dynamic vehicle routing problems (VRP) have always been challenging, and the airborne version of the VRP adds dimensions and difficulties not present in typical ground-based applications. Previous UAV routing work has focused primarily on static, pre-planned situations; however, scheduling military operations, which are often ad-hoc, drives the need for a dynamic route solver that can respond to rapidly evolving problem constraints. With these considerations in mind, we examine the use of a Java-encoded metaheuristic to solve these dynamic routing problems, explore its operation with several general problem classes, and look at the advantages it provides in sample UAV routing problems. The end routine provides routing information for a UAV virtual battlespace simulation and allows dynamic routing of operational missions.

INTRODUCTION

Unmanned Aerial Vehicle (UAV) routing is a complex problem, and earlier work on the subject examined essentially predefined static scenarios. A tabu search coupled with a Monte Carlo Simulation was used to find the minimum number of vehicles required based on stochastic survival probabilities (Sisson 1997). Stochastic simulations involved selecting the best predefined route based on expected values of service, wind, and survival variables (Ryan 1998). This produced a *robust tour* which could then be used to mission plan a given set of targets with unknown threat and wind conditions at the time of mission execution. This approach is wholly appropriate for an autonomous UAV which is preprogrammed to execute a planned mission. While this gives a good starting point for a route schedule, it does not incorporate the latest information—information that can rapidly change.

The continuously evolving mission is a primary concern, especially to the

operators of a long-duration, unmanned aerial vehicle such as the US Air Force's RQ-1A Predator. An ability to dynamically adapt to the latest target update is fundamental to successful military operations. Therefore, we seek to take maximum advantage of current information (winds, target locations, threats, priorities) to dynamically generate and update routes for real-time use. This requires a method fast enough to be operationally effective, robust enough to handle a wide scope of problems, and reliable enough to provide optimal (or near optimal) solutions.

Most routing problems are NP-hard combinatorial problems for which no polynomially bounded algorithm has been found (Bodin et al. 1983). Convergent algorithms can rarely solve large problems consisting of more than 50 customers and often require relatively few side constraints (Gendreau et al. 1997). Unfortunately, real-world problems, such as UAV routing, possess many side constraints such as route and vehicle capacities, route length restrictions, and time windows in a sizeable network. Additionally, this network may be comprised of multiple depots and heterogeneous vehicles. Finding optimal solutions to these types of problems by using techniques such as branch and bound or dynamic programming is currently not practical.

Several heuristic approaches have been used in an attempt to overcome these problems. Greedy algorithms, which prove to be very useful in simpler problems, fail to achieve the desired results with respect to solution quality. Simulated annealing (SA) displays large variances in computational time and solution quality due to the random nature of its search strategy (Osman 1993). Genetic algorithms (GAs), which are designed to solve numerical optimization problems rather than combinatorial optimization problems, are difficult to apply to vehicle routing problems (VRPs) that require capacity, distance, and time window constraints (Gendreau et al. 1997). Fortunately, tabu search (TS) (Glover 1989) provides excellent results on these types of problems. The tabu search heuristic uses adaptive memory structures as it searches the solution space. Moves from one solution to another are made in a forced and orderly manner, and this forced move methodology allows the tabu search to *escape* the local extreme points. At each iteration, the tabu search will select a solution from the neighborhood provided the new candidate solution is not on the *tabu list*. The tabu list is a data structure which keeps track of past solutions visited so that new solutions must be examined. Since the search must pick a new solution at each iteration, the items on the tabu list will be tabu, or off-limits, and the heuristic will pick the best non-tabu move, which may actually be a worse solution. This seems somewhat counter-intuitive, but the search will continue on to find unexplored areas which potentially may yield better overall results. A special instance called *aspiration* allows the tabu status of a move to be overruled if certain conditions are met. The tabu status will be overridden and the solution accepted if it is deemed good enough based on certain attractiveness thresholds. The length of time a solution stays on the tabu list is determined by the tabu list length. Based on the length of the tabu list, the behavior of the search can be significantly altered. If the list is shortened, *intensification* occurs and the local area will be searched more thoroughly as the search gravitates towards the local optimum. If the list is lengthened, *diversification* occurs and the search will be forced leave its current area to explore new areas further away in the solution space (Glover 1997).

The literature shows TS is a robust approach to solving many variations of the

VRP and dominates current studies of routing problems (Garcia et al. 1994, Osman 1993, Rochat and Semet 1994, Carlton 1995, Xu and Kelly 1996, Chiang and Russell 1997, Gendreau et al. 1997, Barbarosoglu and Ozgur 1999). Even certain vehicle routing methods, such as the sweep method and petal heuristic, are not as powerful as tabu search algorithms (Renaud et al. 1996b).

This project explores the application of the reactive tabu search (RTS) metaheuristic to routing problems, specifically the vehicle routing problem with time windows (VRPTW). Our RTS follows the basic TS scheme, but differs in that it actively adjusts the tabu length based on the quality of the search, as determined by the number of iterations before a solution is revisited. In execution this project implements the object-oriented (OO) Java programming language for two reasons. First, the OO design of software allows us to reuse and modify existing code and libraries which reduces the development time of new software routines to extend problems (Eckel 1998). Second, Java programs offer a cross-platform compatibility which enhances portability. Our Java heuristic implementation follows, improves, and extends a MODSIM implementation (Ryan 1998) based on an RTS developed by Battiti and Tecchiolli (1994) and implemented by Carlton (1995).

In this paper, first we examine a reactive tabu search heuristic suitable for solving traveling salesman and vehicle routing problems and provide our results from a Java implementation of this solver. We look at enhancements to the RTS, the verification and validation results, and explore how this tabu search successfully solves tough problems. We review past work and general formulation of the UAV routing problem. We look at our modifications to previous efforts and show how the RTS enables us to solve this problem in particular. Finally, we suggest areas for future exploration.

REACTIVE TABU SEARCH FOR THE VEHICLE ROUTING PROBLEM WITH TIME WINDOWS

The vehicle routing problem with time windows (VRPTW) is defined as follows. Let $G = (V, A)$ be a graph where $V = \{v_0, v_1, \ldots, v_n\}$ is the vertex set and $A = \{(v_i, v_j) : v_i, v_j \in V,\ i \neq j\}$ is the arc set. The depot vertex v_0, has m identical vehicles, each with a maximum load capacity Q and a maximum route duration D. The remaining vertices $v_i \in V$ represent customers to be serviced, each with a non-negative demand q_i, a service time s_i, and a service time window comprised of a no-earlier-than time e_i and a no-later-than time ℓ_i. The no-earlier-than time window constraint is considered soft, i.e., an arrival time a_i before the early time results in a wait time w_i until e_i to commence service. Each edge (v_i, v_j) has an associated non-negative cost c_{ij}, interpreted as travel time t_{ij} between locations i and j.

The objective of the vehicle routing problem with time windows (VRPTW) is to determine a set of m vehicle routes starting and ending at the depot, such that each customer is visited exactly once within its time window, the total demand of any vehicle route does not exceed Q, the duration of any vehicle route does not exceed D, and the total cost of all routes is minimized. When only one vehicle is available and Q, D, e_i, and ℓ_i are non-binding constraints, the problem reduces to a traveling salesman problem (Renaud et al. 1996a).

A *tour* is defined by the order in which the n customers are served by the m vehicles. In our heuristic, we represent the

problem as an ordered list of the sequence of customers and vehicles, or *disjunctive graph*, as shown in Figure 1.

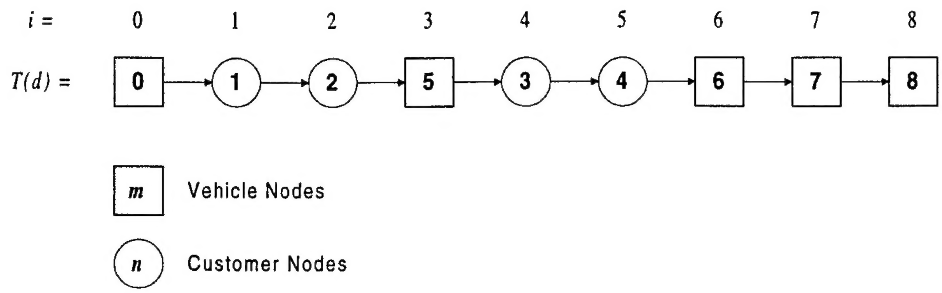


Figure 1. Disjunctive Graph Notation

The first and last positions (*0* and $n + m$) in this sequence represent the initial depot/vehicle and an additional terminal depot required to close the graph. These two nodes are fixed and will not move during the search. Initially, the customers occupy positions between 1 and n and the additional vehicles occupy the remaining positions between $n + 1$ and $n + m - 1$ as shown in Figure 2. During the search, customers and vehicles will be interspersed, and unused vehicles will occupy positions between the last serviced customer and the final depot.

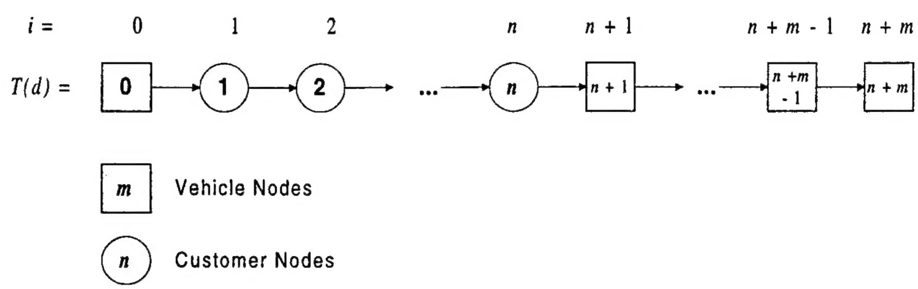


Figure 2. Initial Tour Sequence

OBJECTIVE FUNCTION

For the generic VRPTW, we seek to minimize travel costs c_{ij} along the selected arcs identified by $x_{ij} = 1$. This is given by

$$\text{minimize } Z_f(t) = \sum_j \sum_i c_{ij} x_{ij} \qquad (1)$$

where

$$X = (x_{ij}) \in S, \quad x_{ij} \in \{0,1\} \quad \forall\ i, j.$$

Full enumeration of all constraints is available in Appendix A.

Penalized Objective Function

A major advantage of our method is that it effectively explores the solution space by considering both feasible and infeasible solutions. First, instead of being restricted

only to feasible regions, our RTS can traverse regions of infeasibility to include starting with an infeasible initial solution. Second, the infeasible solutions generated may be used in real world applications with flexible constraints. For instance, an infeasible solution that produces superb overall results may become feasible with the relaxation of a constraint controlled by the decision-maker. Such a case occurred with a delivery problem solved by Rochat and Semet (1994). Since very few real-world constraints are absolutely hard, these infeasible solutions may represent some difficult route selection choices that managers may face when trying to balance competing criteria.

A solution is infeasible if it violates a time window, load capacity, or duration constraint. Constraint violations include missed time windows *TW* and excess vehicle load capacity *LD* defined as

$$TW = \sum_i \left[\max(0, a_i - \ell_i)\right] + \sum_i \left[\max(0, a_i - D_i)\right]$$

and

$$LD = \sum_i \left[\max(0, q_i - Q_i)\right]$$

respectively. Each constraint violation is scaled by a corresponding penalty factor, ρ_{TW} and ρ_{LD}, giving the penalized objective function as

$$\min\ Z(t) = Z_f(t) + \rho_{LD} LD + \rho_{TW} TW \quad (2)$$

where $Z_f(t)$ is the original objective function given by (1). If the solution is feasible, then $Z_f(t)$ and $Z(t)$ are equivalent. Otherwise, $Z(t)$ will include non-zero penalty terms.

Adjusting Reactive Penalty Coefficients

The penalty factors should be large enough to separate the infeasible and feasible regions of the solution space so that infeasible solutions do not dominate feasible solutions. The penalty factors should also be small enough to allow consideration of infeasible solutions. Appropriate penalty values can be very difficult to calculate (Petridis et al. 1998), so our implementation allows for self-adjusting penalty values in addition to constant user-set penalty values.

When self-adjusting, the value of the penalty coefficients ρ_{LD} and ρ_{TW} are independently adjusted every five iterations as proposed by Gendreau et al. (1996) using the relationship

$$\rho_{TW} = \rho_{TW} \cdot 2^{\frac{t_{TW}}{5} - 1}$$

$$\rho_{LD} = \rho_{LD} \cdot 2^{\frac{t_{LD}}{5} - 1}$$

where t_{TW} is the number of time window infeasible solutions among the last ten solutions and t_{LD} is the number of capacity infeasible solutions among the last ten solutions. If all ten previous solutions are feasible, the current ρ is multiplied by $\frac{1}{2}$. If all ten previous solutions are infeasible, the current ρ is multiplied by 2. Intermediate numbers of infeasible solutions yield multiplicative factors between $\frac{1}{2}$ and 2. The penalty values are arbitrarily limited to the closed interval $[0.1, 10^{200}]$, a range easily represented by Java. This prevents the penalties from being rounded by Java to unadjustable zero or infinity values. In the reactive penalty scheme, we arbitrarily set both penalty values initially to 1000.

The reactive penalties provide a measure of trajectory control into and out of feasible regions based on the collective feasibility of the previous solutions. When

many successive solutions are feasible, the lowered penalties do not strongly discourage movement to an infeasible solution. Successive infeasible solutions drive the penalties higher, putting increasingly greater emphasis on finding a feasible solution.

Initial Solution

An initial solution, which may or may not be feasible, is arbitrarily constructed. We employ three options for arranging this initial solution—the first is a listed ordering, the second is based on the time window midpoint, and the third is based on a randomized ordering. All three methods arbitrarily construct a solution by assigning all customers to one vehicle.

The list ordered tour method (LOT) simply assigns customers to the vehicle in the order that they are listed in the data set. The ordered starting tour (OST) method generates a starting solution by sorting the customers based upon increasing time window midpoint values while enforcing the time window feasibility conditions. The time window midpoint for the customer i is defined as halfway between e_i and ℓ_i.

The random starting tour (RST) method randomly reorders the sequential starting list of customers to provide a different starting point. Since the tabu search is a neighborhood search, the initial starting solution will influence the progression of the search. Our experimentation suggests that the reactive tabu search is robust and relatively insensitive to the initial tour.

Neighborhood Structure

Our solution neighborhood is the set of tours immediately reachable from the current solution with a single 3-opt move. The 3-opt move removes three edges and replaces them with three new edges in a way that moves one vertex to another location in the tour sequence. From the disjunctive graph formulation, the solution neighborhood is examined with incremental *swap* moves and updated with an *insertion* move. A swap move exchanges the position of two adjacent nodes with a 3-opt move as shown in Figure 3. An insertion move relocates a specific customer at location i forwards or backwards in the tour by a number of steps called the insertion depth d. In our implementation, an insertion is executed as a series of sequential swap moves.

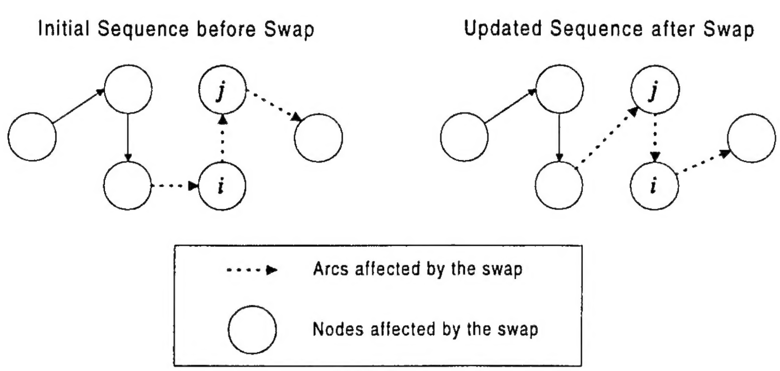


Figure 3. Adjacent 3-Opt Swap Move

This move type yields a staggering $(n + m - 1)!$ possible solution permutations—a relatively simple 25 customer, 5 vehicle problem has 8.842×10^{30} possible solutions. To reduce the neighborhood size, moves which result in a redundant tour are prohibited. Additionally, strong time window feasibility is enforced (Carlton 1995).

Strong time window infeasible states occur between nodes i and j whenever a vehicle leaving node i at departure time d_i can never arrive at node j within the required time window. Specifically, node j is strong time window infeasible with respect to node i if

$$d_i + t_{ij} > \ell_j \;\; \forall \; d_i = a_i + s_i, \; a_i \in [e_i, \ell_i].$$

Weak time window infeasible states occur when only some departure times preclude a timely arrival at the following node, i.e.,

$$\begin{gathered} d_i + c_{ij} \leq \ell_j \\ \forall \, d_i = a_i + s_i, \\ a_i \leq t, t \in [e_i, \ell_i). \end{gathered}$$

Unlike strong time window infeasible tours, weak time window infeasible tours are evaluated in the search since insertion moves can ultimately reduce the amount of infeasibility in the overall tour (Carlton 1995). Past vehicle routing problem research indicates that feasible solutions may be isolated or disjoint from each other in the solution space, so in order to effectively search the solution space, the method must investigate and perhaps accept infeasible solutions. This search of the infeasible region is facilitated by our use of penalty factors.

Tabu Moves

Tabu search uses a memory structure to determine if a particular tour has already been visited by examining its attributes. The examination must efficiently and reliably store and identify solution attributes previously altered during the search. We employ an $(n+1) \times (n+1)$ dimension *Tabulist* matrix with rows corresponding to customer identification numbers and columns corresponding to the index, or position, of the customer in the solution tour. The data elements in this array store the iteration number k for the move that placed the customer into this position plus the tabu length θ. This value will be compared to the current iteration to determine if a move of this attribute is tabu.

Adjusting Tabu Length

To maintain search quality, we reactively adjust the tabu length based on the number of iterations occurring between *cycles*. Cycles occur when the search revisits a solution; a high quality search should infrequently revisit past solutions. Given the combinatorial nature of the problem, it is possible to select a seemingly different tour that is actually a *redundant tour*—one that appears new, but in fact is a revisit of a previous equivalent tour. Figure 4 illustrates two different tours which are actually redundant tours.

Redundant tours are identified with a two-attribute hashing scheme. The first hashing attribute, the *hashing function* $f(t)$, is assigned the objective function value $Z(t)$. Woodruff and Zemel (1993) propose a method that we use to compute the second hashing attribute, the *tour hashing value* $thv(t)$. We take the tour vector and calculate an integer based on random integer values, $\Psi(\tau_i)$, where τ_i is the index of the customer assigned to tour position i, such that

$$thv(T) = \sum_{i=0}^{n} \Psi(\tau_i) \cdot \Psi(\tau_{i+1}) \ .$$

This tour hashing value attempts to minimize the occurrence of a *collision*, or the incorrect identification of two tours as being identical or redundant when they are actually distinct.

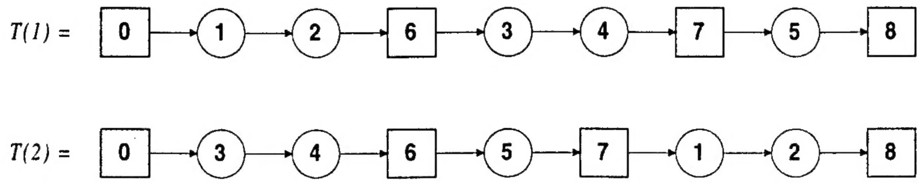


Figure 4. Redundant Tours

We also use other attributes to identify a solution; these are tour cost, travel time, time window penalty, and total penalty. These integer values are concatenated into a uniquely identifiable Java string object and stored with Java Hashtable class functions. This unique string value allows us to efficiently identify past solutions, as well as access the hash record containing solution attributes stored in their original form.

When the search revisits a solution within the designated number of iterations, or *cycle length*, the tabu length is increased by a scaling factor. This tabu length increase diversifies the search. If the search is not revisiting solutions, tabu length is decreased by a scaling factor. When a solution is revisited within the *maximum cycle length*, the algorithm calculates a moving average of cycle lengths, or the average number of iterations between a revisit. If the tabu length has remained unchanged for a number of iterations greater than or equal to this moving average, then the current tabu length is decreased by the scaling factor, thus intensifying the search. We set the initial tabu length value θ to the smaller of either 30 or $m + n - 1$.

Aspiration and Escape Functions

Aspiration allows for overriding the tabu status of a move if the proposed tour solution is better than any previous solution. If all moves are tabu and no proposed solution meets aspiration criteria, the search *escapes* to the neighbor tour with smallest move value. This escape move is accomplished regardless of tabu status and results in a tabu length decrease.

Move Evaluation and Selection

The RTS systematically explores the solution space using a series of swap moves and chooses the allowable adjacent solution with the smallest move value. The move value is the difference between the incumbent's objective function value and the candidate's objective function value given as the cost/travel savings resulting directly from the 3-opt move and the resultant changes occurring in the rest of the tour.

Heuristic Description

Crucial to the success of the solver is the *time matrix* which contains the travel times t_{ij} between every node combination i,

j. The time matrix is built in a three-step process. First, cartesian distances between locations are computed. Second, these distances are converted to times based on problem parameters. Third, the service time at node i is added to the time. As such, t_{ij} values then represent the amount of time between arrival at node i and the subsequent arrival at node j. We use these values as our costs, i.e., $c_{ij} = t_{ij}$. Actual en route travel time can be calculated by subtracting service time s_i from t_{ij}.

The reactive tabu search executes the following steps:

Step 1 (Initialization) Initialize data structures, vectors, and parameters.

Step 2 (Problem Input) Read data and assign node information. Calculate appropriate time matrix.

Step 3 (Route Initialization) Construct initial tour, calculate initial tour schedule, and compute associated tour cost and hashing value. Store values. Assign initial tour as incumbent tour.

Step 4 (Cycle Check) Check hashing structure for the incumbent tour. If found, update the iteration when found, increase the tabu length if applicable. If not found, add to the hashing structure, decrease the tabu length, if applicable. Increment current iteration number.

Step 5 (Check Later Insertions) Accomplish swap moves to evaluate all forwards insertions. Store position i and depth d of best move value, aspiration, and escape information.

Step 6 (Check Earlier Insertions) Accomplish swap move to evaluate all backwards insertions. Store i, d of best move value, aspiration, and escape information.

Step 7 (Execute Move) Move to a non-tabu neighbor according to appropriate decision criteria. If all moves are tabu, use the escape move and reduce the tabu length. Perform insertion, update schedule, assign neighbor tour as new incumbent tour, compute hashing value, and track best tour information. If current iteration number is less than the maximum iteration number, return to Step 4.

Step 8 (Output results) Terminate heuristic search and output results.

Computational Complexity

The neighborhood size considered at each step is $O(nd)$, and the computation of the move value for each neighbor is $O(n)$. If the depth of the insertion moves is restricted to 1, then the algorithm achieves a minimum computational complexity of $O(n^2)$. The worst case complexity is $O(n^2d)$ where d is the depth of the allowable insertion moves. When the insertion depth is expanded to n the computational complexity expands with it to a maximum $O(n^3)$. However, empirical testing shows that considerably better times than $O(n^3)$ can be achieved due to the strong time window infeasibility restriction discussed earlier (Carlton 1995).

REACTIVE TABU SEARCH FOR DYNAMIC UNMANNED AERIAL VEHICLE ROUTING

The US Air Force uses the Predator UAV to perform a reconnaissance and surveillance mission. The Predator is remotely flown by Air Vehicle Operators, who are Air Force pilots, located in a Ground Control Station. Co-located Payload Specialists remotely control the electro-optical camera, infrared scanner, and synthetic aperture radar to observe targets of interest as specified by higher command elements. The imagery is returned real-time via satellite link to intelligence specialists and regional commanders (McKenna 1998). The Predator has been used successfully to monitor buildings, military forces, and battle activities in Bosnia pursuant to United Nations and NATO missions. The Predator's long airborne endurance of nearly 40 hours and its ground based control system (with ready access to computers) makes it an ideal candidate for efficient computerized routing strategies.

We seek to enhance the capabilities of existing mission software. Current software will automatically generate *deterministic* items such as terrain avoidance profiles, ground station to UAV line of site availability, route times between defined way points, fuel consumption, heading and turn information, etc., but it does not and will not optimize routes. This *combinatorial* problem is a task left to the operator. We provide our routing tool to fill the gap that exists in making complicated routing decisions.

Since this is a real-world operational problem, several real-world operational factors influence our implementation approach.

Operational Parameters

Operational employment of the UAV drives several changes to how the problem data is specified and solved. These changes range from relatively superficial ones in how the coordinates and times are represented, to moderate changes in how the parameters are calculated, to significant changes in how the objective function is formulated to reflect the nature of the problem.

Geographic Coordinates

Coordinates are expressed in a geocentric format instead of a Cartesian format. We calculate the distance and bearing between coordinate points as shown in AFR 51-40, Air Navigation (Departments of the Air Force and Navy 1983). Given the departure latitude L_1 and longitude λ_1 and the destination latitude L_2 and longitude λ_2, the great circle distance d in nautical miles between the two coordinate points can be found using the following formulation

$$d = 60 \cdot \cos^{-1}\left[\sin L_1 \cdot \sin L_2 + \cos L_1 \cdot \cos L_2 \cdot \cos(\lambda_2 - \lambda_1)\right].$$

Using this distance, an intermediate heading angle H in degrees is determined as

$$H = \cos^{-1}\left[\frac{\sin L_2 - \sin L_1 \cdot \cos\left(\frac{d}{60}\right)}{\sin\left(\frac{d}{60}\right) \cdot \cos L_1}\right].$$

Based on the geometry of the coordinates, this intermediate heading angle is adjusted to obtain the initial true heading Θ_{ij}, measured in degrees from true north, i.e.,

$$\Theta_{ij} = \begin{cases} H, & \sin(\lambda_2 - \lambda_1) < 0 \\ 360^\circ - H, & \sin(\lambda_2 - \lambda_1) \geq 0 \end{cases}.$$

This distance and bearing geometry is shown in Figure 5.

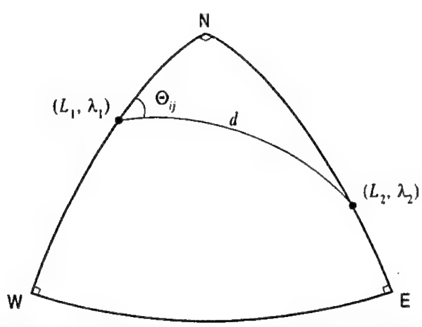


Figure 5. Distance and Bearing Geometry (Spherical Triangle)

Wind Effects on Ground Speed

When computing transit times between locations, we must account for the effect of winds aloft. Given a wind speed WS from a bearing of Θ_{WS} measured in degrees from true north, one can calculate the effective ground speed GS along the true course Θ_{ij} from the first location to the second. The difference between Θ_{ij} and Θ_{WS} is represented by δ. Figure 6 illustrates this geometry. When $|\delta| < 90$, A is negative and subtracts from the airspeed as a headwind component. When $90 < |\delta| \leq 180$, A is positive and adds to the airspeed as a tailwind component. The wind correction angle from true heading is denoted by γ. This adjusted heading corrects the flight path to compensate for wind drift. Groundspeed, as influenced by wind aloft, is explicitly calculated as

$$\begin{aligned} \delta &= \Theta_{ij} - \Theta_{WS} \\ A &= WS \cdot \cos(180 - \delta) \\ C &= WS \cdot \sin(180 - \delta) \\ B &= \sqrt{AS^2 - C^2} \end{aligned}$$

and

$$GS = A + B = WS \cdot \cos(180 - \delta) + \sqrt{AS^2 - WS^2 \cdot \sin^2(180 - \delta)}.$$

The transit time between the points is then simply $t_{ij} = d_{ij}/GS$.

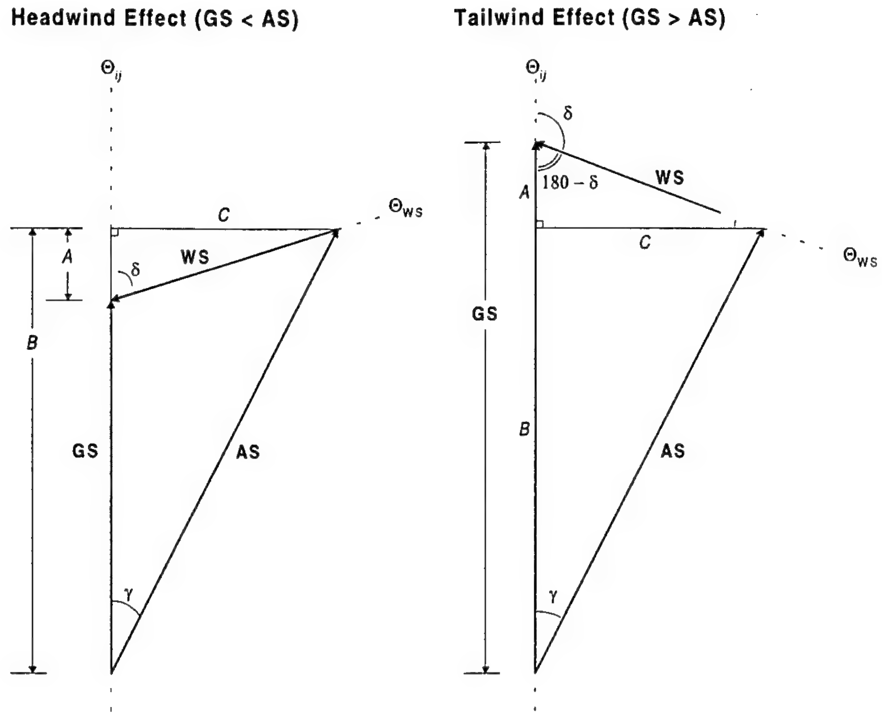


Figure 6. Headwind and Tailwind Ground Speed Adjustment

Numerical Formatting

The latitude L and longitude λ information is measured in degrees where one degree is composed of sixty minutes and one minute is composed of sixty seconds. The are values often listed in a *degrees minutes seconds* format (*DD MM SS.ss*); we convert latitudes and longitudes into a decimal degree format for computational ease using the formula

$$D.d = DD + MM/60 + SS.ss/3600.$$

Locations can also be listed in a *degrees minutes decimal minutes* format (*DD MM.mm*) where minutes are expressed as decimal values. Conversion to a decimal degree value is defined as

$$D.d = DD + MM.mm/60.$$

Clock time is often expressed in a military-style *hours minutes* (*HH MM*) format; for computational ease, we express time in minutes $t_{minutes}$ as

$$t_{minutes} = 60 \cdot HH + MM.$$

Objective Function Modifications

The UAV operating environment also mandates changes to the objective function. The standard VRPTW objective function seeks to minimize travel costs as

represented by the distance traveled. Early arrival to a customer is allowed, and the resulting waiting time is cost free in the objective function. This may be appropriate for a standard terrestrial application in which the costs are associated mainly with transiting between locations, but in UAV operations there are costs associated with keeping the aircraft airborne. Thus, UAV waiting times represent costs that must be considered in our efforts to minimize the objective function. We therefore modify the original and penalized objective functions (1) and (2) to include waiting time w_i at node i in addition to the original transit times as

$$\text{minimize } Z_f'(t)=\sum_j\sum_i(c_{ij}+w_i)x_{ij} .$$

As before, the penalized objective function is gained by adding the scaled infeasibility values to yield

$$\text{minimize } Z'(t)=Z_f'(t)+\rho_{LD}LD+\rho_{TW}TW .$$

The search now attempts to minimize the total time aloft and proceeds as previously presented.

Dynamic Mission Requirements

The nature of UAV employment presents unique situations that our routing tool must handle. As such, we show how our scenarios depart from traditional VRPs and explain how we successfully implement these requirements. Unique routing situations exist with regard to *altitude-based wind tiers*, random service times, emerging priority targets, and locked route sequences. These instances are explored in the following paragraphs.

Optimizing Use of Altitude-Based Wind Tiers

Recall that in the general MTSPTW problem, travel times between fixed locations are known, with fixed and symmetric costs (i.e., $c_{ij}=c_{ji}$). This symmetry does not hold in the UAV operating environment where winds affect travel times and can vary both in direction and velocity as a function of altitude (depicted below in Figure 7). We incorporate this wind information to select the minimum travel time between nodes.

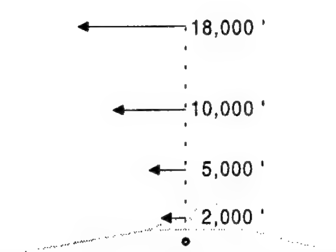


Figure 7. Typical Winds Aloft Profile

Specifically in any altitude band k, travel time is a function of UAV altitude h_{ijk} and airspeed AS_k; wind speed WS_k and direction Θ_{WSk}; and the distance d_{ij} and bearing Θ_{ij} between locations. We make the simplifying assumption that wind

direction and speed is constant throughout an entire altitude zone. This is reasonable since values at any point in the region are interpolated predictions based on measurements of actual conditions at discrete weather station locations (Parsons 1999).

Using our previous equations, we calculate times between all locations based on the adjusted ground speed for each altitude band. This forms tiers of asymmetric wind-influenced travel time matrices from which we select the smallest travel time from i to j as

$$t_{ij} = \min_{\forall k} \left[\frac{d_{ij}}{GS_{ij}(h_{ijk})} \right]$$

where $GS_{ij}(h_{ijk})$ the ground speed as a function of traveling in altitude band k. The corresponding altitude is assigned as our flight altitude for that leg. Since this wind optimization process is accomplished prior to beginning the tabu search, the heuristic will accept an arbitrary number of altitude bands with no appreciable effect on computational time or efficiency.

Random Service Times

In the general TSPTW problem, customer service times are known constants. In the UAV problem context, the target service times are random variables. The service time represents the amount of time the aircraft spends circumnavigating the target point to gather imagery from multiple viewpoints, and, due to the unknown nature of the target, military necessity may dictate a longer observation than initially planned. The actual target i service time S_i falls between the minimum service time $s_{min}(i)$ and the maximum service time $s_{max}(i)$ inclusive. Service time will be the minimum service time with 0.7 probability; when the time is above the minimum, it is modeled as uniformly distributed between the minimum and maximum. The service time is given by

$$S_i = \begin{cases} s_{min}(i) & \text{with 0.7 probability} \\ \text{Uniform}(s_{min}(i), s_{max}(i)) & \text{with 0.3 probability} \end{cases}.$$

A known service time is simply specified by setting $S_i = s_{min}(i) = s_{max}(i)$.

Emerging Targets

Another aspect of UAV operations is the pop-up priority target. This occurs when the UAV is retasked in flight to observe a target of utmost military urgency. Depending on the new target location, this immediate divert may render the remainder of the route obsolete. Rather than proceed with a potentially sub-optimal route, our solver offers the ability to *route-from-here*.

Given that the UAV will proceed to the ad hoc target, this location becomes a new starting point and the remaining targets are processed in a route that returns the UAV to the depot. This route-from-here capability is achieved with smart processing of the time matrix.

Locked and Forbidden Routes

At times, UAV operations require a *locked route*, in which one or more targets must be visited in a specific order. This may occur with a directed route or with certain observational requirements such as a consecutive imaging pass for a synthetic aperture radar image. The GUI allows these route points to be locked together and treated as an *aggregated node* with a beginning location corresponding to the first point and an ending location corresponding to the last point. The aggregated node is assigned a composite service time that accounts for intra-node service, wait, and travel times.

The opposite of a locked route is a *forbidden route* which may be a result of a no-fly zone or threat region. The forbidden area is then monitored for flight paths which pass through it; if a path intersects a forbidden area, it is modeled as a longer route that skirts the edge of the region as shown in Figure 8.

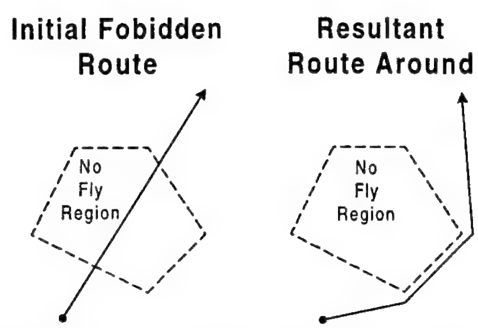


Figure 8. Forbidden Route Example

COMPUTATIONAL RESULTS

General Results

Our initial testing and validation used the Solomon VRPTW problem test sets—25, 50, and 100 customer scenarios with random, clustered and random clustered distribution patterns. Our computational results are compared in Tables 1-6 (Ryer 1999) to known optimal answers obtained by Desrochers, Desrosiers, and Solomon (1992). Dashed regions of the chart indicate problems that could not be optimally solved by Desrochers et al. All problems were solved in reasonable computation times by our RTS algorithm (2500 iterations with user specified penalties) with an overall solution quality within 1% of optimal values. Solving the harder VRPTW class problems did not require an increase in computation times over the mTSPTW class problems.

The objective function value used in these initial tests includes travel time, missed time window penalties, and load overage penalties. With a relatively small amount of coding, the objective function can be expanded to include additional penalties, changed to represent several different weighted objective functions, or combined in a hierarchical objective function. Results are presented in Tables 1 through 6.

Table 1. Solomon mTSPTW Computational Results (25 Customers)

Problem Set[1]	O'Rourke & Ryer $Z_t(t)$	Used	Iter[2]	Time[3]	Optimal $Z_t(t)$	Used	Time[4]	Difference Δ	Δ%	Start Method[5]
R101	867.1	8	317	3	867.1	8	5.8	0.0	0.00%	OST
R102	797.1	7	35	1	797.1	7	20.3	0.0	0.00%	OST
R103	704.6	5	132	1	704.6	5	22.2	0.0	0.00%	OST
R104	666.9	4	86	1	666.9	4	46.0	0.0	0.00%	OST
R105	780.5	6	95	1	780.5	6	22.6	0.0	0.00%	OST
R106	715.4	5	28	0	715.4	5	205.2	0.0	0.00%	RST 0
R107	674.3	4	2080	23	674.3	4	304.1	0.0	0.00%	RST 2
R108	647.3	4	45	0	647.3	4	307.4	0.0	0.00%	OST
R109	691.3	5	21	0	691.3	5	14.4	0.0	0.00%	OST
R110	694.1	5	91	2	679.8	4	64.3	14.3	2.10%	RST 0
R111	678.8	4	178	2	678.8	4	330.3	0.0	0.00%	RST 0
R112	643.0	4	25	0	643.0	4	623.3	0.0	0.00%	LOT
C101	2441.3	3	23	0	2441.3	3	18.6	0.0	0.00%	OST
C102	2440.3	3	379	4	2440.3	3	79.9	0.0	0.00%	LOT
C103	2440.3	3	72	1	2440.3	3	134.7	0.0	0.00%	OST
C104	2436.9	3	797	8	2436.9	3	223.9	0.0	0.00%	OST
C105	2441.3	3	209	2	2441.3	3	25.6	0.0	0.00%	OST
C106	2441.3	3	26	1	2441.3	3	20.7	0.0	0.00%	OST
C107	2441.3	3	28	1	2441.3	3	31.7	0.0	0.00%	OST
C108	2441.3	3	1421	15	2441.3	3	43.1	0.0	0.00%	OST
C109	2441.3	3	148	1	2441.3	3	585.4	0.0	0.00%	OST
RC101	711.1	4	214	3	711.1	4	225.4	0.0	0.00%	LOT
RC102	601.7	3	20	1	596.0	3	18.1	5.7	0.96%	OST
RC103	582.8	3	2193	24	582.8	3	103.0	0.0	0.00%	RST 2
RC104	556.6	3	604	6	556.6	3	177.9	0.0	0.00%	OST
RC105	661.2	4	79	1	661.2	4	37.4	0.0	0.00%	RST 1
RC106	595.5	3	60	1	595.5	3	248.4	0.0	0.00%	RST 1
RC107	548.3	3	69	1	548.3	3	113.9	0.0	0.00%	RST 0
RC108	544.5	3	2203	23	544.5	3	256.0	0.0	0.00%	OST
Average	1218.19	3.93	402.7	4.38	1184.8	3.90	148.6	0.69	0.11%	—

(Ryer 1999)

[1] Maximum number of vehicles: $m = 10$. Time window penalty: $\rho_{TW} = 1.0$.
[2] Maximum iterations: $k = 2500$.
[3] Seconds on a Pentium II 400 MHz system. Total runtime ~ 28 seconds each.
[4] Seconds on a Sun Sparc 1 workstation.
[5] OST is ordered starting tour. RST is random starting tour seeded with the value given. LOT is listed ordering.

Table 2. Solomon mTSPTW Computational Results (50 Customers)

Problem Set[1]	O'Rourke & Ryer $Z_i(t)$	O'Rourke & Ryer Used	O'Rourke & Ryer Iter[2]	O'Rourke & Ryer Time[3]	Optimal $Z_i(t)$	Optimal Used	Optimal Time[4]	Difference Δ	Difference Δ%	Start Method[5]
R101	1543.8	12	239	9	1535.2	12	66.7	8.6	0.56%	RST 0
R102	1409.0	11	1939	78	1404.6	11	67.8	4.4	0.31%	RST 0
R103	1282.7	9	871	36	1272.5	9	8939.1	10.2	0.80%	OST
R104	1131.9	6	734	31	—	—	—	—	—	RST 0
R105	1401.6	9	402	15	1399.2	9	362.6	2.4	0.17%	LOT
R106	1293.0	8	2294	94	1285.2	8	386.4	7.8	0.61%	RST 1
R107	1211.1	7	1786	75	1211.1	7	7362.1	0.0	0.00%	RST 0
R108	1117.7	6	1698	75	—	—	—	—	—	RST 0
R109	1286.7	8	1452	58	—	—	—	—	—	RST 0
R110	1207.8	7	1853	78	1197.0	7	4906.1	10.8	0.90%	RST 1
R111	1216.6	7	1775	72	—	—	—	—	—	RST 2
R112	1140.5	6	1784	78	—	—	—	—	—	RST 2
C101	4862.4	5	119	4	4862.4	5	67.1	0.0	0.00%	LOT
C102	4861.4	5	607	19	4861.4	5	330.2	0.0	0.00%	LOT
C103	4855.8	5	1699	57	—	—	—	—	—	OST
C104	4884.1	5	1253	43	—	—	—	—	—	LOT
C105	4861.2	5	232	7	—	—	—	—	—	OST
C106	4862.4	5	308	9	4862.4	5	91.3	0.0	0.00%	LOT
C107	4861.2	5	382	12	—	—	—	—	—	LOT
C108	4861.2	5	92	3	—	—	—	—	—	LOT
C109	4860.9	5	301	9	—	—	—	—	—	OST
RC101	1444.0	8	1252	38	—	—	—	—	—	RST 1
RC102	1325.1	7	754	23	—	—	—	—	—	RST 1
RC103	1216.2	6	1589	54	—	—	—	—	—	RST 0
RC104	1046.5	5	860	31	—	—	—	—	—	RST 2
RC105	1355.3	8	248	8	—	—	—	—	—	OST
RC106	1223.2	6	1921	61	—	—	—	—	—	RST 2
RC107	1146.0	6	189	7	—	—	—	—	—	LOT
RC108	1098.1	6	1821	65	—	—	—	—	—	OST
Average	2374.7	6.66	1050	39.6	—	—	—	—	—	—

(Ryer 1999)

[1] Maximum number of vehicles: R sets $m = 15$; C sets $m = 6$; RC sets $m = 8$. Time window penalty: $\rho_{TW} = 3.0$.
[2] Maximum iterations: $k = 2500$.
[3] Seconds on a Pentium II 400 MHz system. Total runtime ~ 100 seconds each.
[4] Seconds on a Sun Sparc 1 workstation.
[5] OST is ordered starting tour. RST is random starting tour seeded with the value given. LOT is listed ordering.

Table 3. Solomon mTSPTW Computational Results (100 Customers)

Problem Set[1]	O'Rourke & Ryer $Z_r(t)$	Used	Iter[2]	Time[3]	Optimal $Z_r(t)$	Used	Time[4]	Difference Δ	Δ%	Start Method[5]
R101	2689.6	20	2167	371	2607.7	18	1064.2	81.9	3.14%	RST 0
R102	2522.9	18	1783	322	2434.0	17	756.9	88.9	3.65%	RST 0
R103	2266.8	15	1797	351	—	—	—	—	—	RST 2
R104	2010.6	11	1401	311	—	—	—	—	—	RST 2
R105	2418.0	16	560	93	—	—	—	—	—	RST 1
R106	2256.9	14	1403	252	—	—	—	—	—	LOT
R107	2091.6	12	1462	278	—	—	—	—	—	LOT
R108	1980.3	10	2325	491	—	—	—	—	—	RST 0
R109	2191.4	13	2149	398	—	—	—	—	—	RST 1
R110	2121.1	12	1479	291	—	—	—	—	—	RST 2
R111	2082.1	12	1882	370	—	—	—	—	—	RST 2
R112	1986.1	11	2325	507	—	—	—	—	—	RST 1
C101	9827.3	10	285	45	9827.3	10	434.5	0.0	0.00%	OST
C102	9820.3	10	237	42	—	—	—	—	—	OST
C103	9813.7	10	256	49	—	—	—	—	—	OST
C104	9809.0	10	2495	536	—	—	—	—	—	RST 2
C105	9821.2	10	313	50	—	—	—	—	—	OST
C106	9827.3	10	455	75	9827.3	10	724.8	0.0	0.00%	OST
C107	9818.9	10	292	48	—	—	—	—	—	OST
C108	9818.9	10	662	115	—	—	—	—	—	OST
C109	9818.6	10	1381	262	—	—	—	—	—	LOT
RC101	2685.7	16	897	144	—	—	—	—	—	OST
RC102	2534.0	15	2410	434	—	—	—	—	—	OST
RC103	2352.3	13	1047	195	—	—	—	—	—	RST 0
RC104	2209.1	11	1311	272	—	—	—	—	—	RST 2
RC105	2538.0	15	2327	412	—	—	—	—	—	RST 1
RC106	2457.8	14	443	74	—	—	—	—	—	RST 0
RC107	2236.9	12	1822	344	—	—	—	—	—	RST 0
RC108	2115.9	11	2206	451	—	—	—	—	—	RST 1
Average	4624.9	12.45	1365	261.48	—	—	—	—	—	—

(Ryer 1999)

[1] Maximum number of vehicles: $m = 25$. Time window penalty: $\rho_{TW} = 8.0$.
[2] Maximum iterations: $k = 2500$.
[3] Seconds on a Pentium II 400 MHz system. Total runtime ~ 550 seconds each.
[4] Seconds on a Sun Sparc 1 workstation.
[5] OST is ordered starting tour. RST is random starting tour seeded with the value given. LOT is listed ordering.

Table 4. Solomon VRPTW Computational Results (25 Customers)

Problem Set[1]	O'Rourke & Ryer $Z_i(t)$	Used	Iter[2]	Time[3]	Optimal $Z_i(t)$	Used	Time[4]	Difference Δ	Δ%	Start Method[5]
R101	867.1	8	317	4	867.1	8	5.8	0.0	0.00%	OST
R102	797.1	7	35	1	797.1	7	20.3	0.0	0.00%	OST
R103	704.6	5	132	1	704.6	5	22.2	0.0	0.00%	OST
R104	666.9	4	86	2	666.9	4	46.0	0.0	0.00%	OST
R105	780.5	6	95	1	780.5	6	22.6	0.0	0.00%	OST
R106	715.4	5	1149	12	715.4	5	205.2	0.0	0.00%	RST 0
R107	674.3	4	2080	24	674.3	4	304.1	0.0	0.00%	RST 2
R108	647.3	4	58	1	647.3	4	307.4	0.0	0.00%	OST
R109	691.3	5	32	1	691.3	5	14.4	0.0	0.00%	OST
R110	694.1	5	91	1	679.8	4	64.3	14.3	2.10%	RST 0
R111	678.8	4	178	3	678.8	4	330	0.0	0.00%	RST 0
R112	643.0	4	25	1	643.0	4	623.3	0.0	0.00%	LOT
C101	2441.3	3	23	0	2441.3	3	18.6	0.0	0.00%	OST
C102	2440.3	3	106	1	2440.3	3	79.9	0.0	0.00%	LOT
C103	2440.3	3	72	1	2440.3	3	134.7	0.0	0.00%	OST
C104	2436.9	3	741	8	2436.9	3	223.9	0.0	0.00%	OST
C105	2441.3	3	170	1	2441.3	3	25.6	0.0	0.00%	OST
C106	2441.3	3	35	1	2441.3	3	20.7	0.0	0.00%	OST
C107	2441.3	3	51	0	2441.3	3	31.7	0.0	0.00%	OST
C108	2441.3	3	455	4	2441.3	3	43.1	0.0	0.00%	OST
C109	2441.3	3	197	2	2441.3	3	585.4	0.0	0.00%	OST
RC101	711.1	4	214	2	711.1	4	225.4	0.0	0.00%	LOT
RC102	601.7	3	149	1	596.0	3	18.1	5.7	0.96%	OST
RC103	582.8	3	134	2	582.8	3	103.0	0.0	0.00%	RST 2
RC104	556.6	3	29	1	556.6	3	177.9	0.0	0.00%	LOT
RC105	661.2	4	24	1	661.2	4	37.4	0.0	0.00%	RST 1
RC106	595.5	3	60	1	595.5	3	248.4	0.0	0.00%	RST 1
RC107	548.3	3	179	2	548.3	3	113.9	0.0	0.00%	RST 1
RC108	544.5	3	353	3	544.5	3	256.0	0.0	0.00%	LOT
Average	1218.2	3.93	250.7	2.86	1184.8	3.90	148.6	0.69	0.11%	LOT

(Ryer 1999)

[1] Maximum number of vehicles: $m = 10$. Time window penalty: $\rho_{TW} = 8.0$; load penalty $\rho_{LD} = 10.0$.
[2] Maximum iterations: $k = 2500$.
[3] Seconds on a Pentium II 400 MHz system. Total runtime ~ 28 seconds each.
[4] Seconds on a Sun Sparc 1 workstation.
[5] OST is ordered starting tour. RST is random starting tour seeded with the value given. LOT is listed ordering.

Table 5. Solomon VRPTW Computational Results (50 Customers)

Problem	O'Rourke & Ryer				Optimal			Difference		Start
Set[1]	$Z_l(t)$	Used	Iter[2]	Time[3]	$Z_l(t)$	Used	Time[4]	Δ	Δ%	Method[5]
R101	1543.8	12	239	9	1535.2	12	66.7	8.6	0.56%	RST 0
R102	1409.0	11	1939	82	1404.6	11	67.8	4.4	0.31%	RST 0
R103	1278.7	9	1935	87	1272.5	9	8939.1	6.2	0.49%	OST
R104	1137.4	6	1533	69	—	—	—	—	—	RST 2
R105	1401.6	9	402	16	1399.2	9	362.6	2.4	0.17%	LOT
R106	1293.0	8	2294	99	1285.2	8	386.4	7.8	0.61%	RST 1
R107	1211.1	7	1786	79	1211.1	7	7362.1	0.0	0.00%	RST 0
R108	1117.7	6	1698	78	—	—	—	—	—	RST 0
R109	1286.7	8	1451	61	—	—	—	—	—	RST 0
R110	1207.8	7	1853	84	1197.0	7	4906.1	10.8	0.90%	RST 1
R111	1216.6	7	1775	76	—	—	—	—	—	RST 2
R112	1135.0	6	1456	68	—	—	—	—	—	RST 2
C101	4862.4	5	74	3	4862.4	5	67.1	0.0	0.00%	LOT
C102	4861.4	5	232	9	4861.4	5	330.2	0.0	0.00%	LOT
C103	4861.4	5	2035	87	4861.4	5	896.0	0.0	0.00%	RST 0
C104	4882.8	5	1727	79	—	—	—	—	—	RST 0
C105	4862.4	5	494	19	4862.4	5	99.1	0.0	0.00%	OST
C106	4862.4	5	91	4	4862.4	5	91.3	0.0	0.00%	LOT
C107	4862.4	5	154	6	4862.4	5	170.6	0.0	0.00%	LOT
C108	4862.4	5	95	4	4862.4	5	245.6	0.0	0.00%	LOT
C109	4862.4	5	643	26	—	—	—	—	—	OST
RC101	1446.8	8	1613	60	—	—	—	—	—	OST
RC102	1331.8	7	1508	60	—	—	—	—	—	RST 2
RC103	1210.9	6	2194	94	—	—	—	—	—	OST
RC104	1046.5	5	412	18	—	—	—	—	—	LOT
RC105	1355.3	8	104	4	—	—	—	—	—	OST
RC106	1223.2	6	1454	58	—	—	—	—	—	RST 2
RC107	1144.4	6	898	36	—	—	—	—	—	RST 1
RC108	1098.1	6	1361	58	—	—	—	—	—	OST
Average	2375.01	6.66	1153	49.4	—	—	—	—	—	—

(Ryer 1999)

[1] Maximum number of vehicles: $m = 15$. Time window penalty: $\rho_{TW} = 1.0$; load penalty $\rho_{LD} = 10.0$.
[2] Maximum iterations $k = 2500$.
[3] Seconds on a Pentium II 400 MHz system. Total runtime ~ 100 seconds each.
[4] Seconds on a Sun Sparc 1 workstation.
[5] OST is ordered starting tour. RST is random starting tour seeded with the value given. LOT is listed ordering.

Table 6. Solomon VRPTW Computational Results (100 Customers)

Problem	O'Rourke & Ryer				Optimal			Difference		Start
Set[1]	$Z_i(t)$	Used	Iter[2]	Time[3]	$Z_i(t)$	Used	Time[4]	Δ	Δ%	Method[5]
R101	2676.2	20	2271	414	2607.7	18	1064.2	68.5	2.63%	RST 2
R102	2502.4	19	492	96	2434.0	17	756.9	68.4	2.81%	RST 0
R103	2265.0	15	1091	228	—	—	—	—	—	RST 2
R104	2039.6	12	1488	338	—	—	—	—	—	OST
R105	2399.4	16	1974	378	—	—	—	—	—	RST 0
R106	2268.4	14	2431	491	—	—	—	—	—	LOT
R107	2129.0	13	1905	406	—	—	—	—	—	RST 1
R108	1956.8	10	2415	565	—	—	—	—	—	RST 0
R109	2181.0	14	1587	311	—	—	—	—	—	RST 1
R110	2133.2	13	1548	328	—	—	—	—	—	RST 2
R111	2077.3	12	2248	491	—	—	—	—	—	RST 2
R112	1971.6	11	1898	460	—	—	—	—	—	RST 2
C101	9827.3	10	263	43	9827.3	10	434.5	0.0	0.00%	OST
C102	9827.3	10	1317	253	9827.3	10	1990.8	0.0	0.00%	OST
C103	9828.9	10	2500	535	—	—	—	—	—	RST 0
C104	9949.6	10	2194	509	—	—	—	—	—	RST 2
C105	9827.3	10	378	65	—	—	—	—	—	OST
C106	9827.3	10	309	55	9827.3	10	724.8	0.0	0.00%	OST
C107	9827.3	10	1144	210	9827.3	10	1010.4	0.0	0.00%	OST
C108	9827.3	10	1638	321	9827.3	10	1613.6	0.0	0.00%	OST
C109	9853.3	10	2202	463	—	—	—	—	—	RST 0
RC101	2669.9	16	2110	381	—	—	—	—	—	OST
RC102	2498.4	15	2136	419	—	—	—	—	—	LOT
RC103	2363.6	13	1333	270	—	—	—	—	—	RST 1
RC104	2179.2	11	1365	308	—	—	—	—	—	LOT
RC105	2557.4	15	2482	473	—	—	—	—	—	OST
RC106	2432.8	13	2222	434	—	—	—	—	—	RST 2
RC107	2266.1	12	2024	417	—	—	—	—	—	RST 2
RC108	2175.1	12	2122	475	—	—	—	—	—	RST 1
Average	4632.3	12.62	1693	349.6	—	—	—	—	—	—

(Ryer 1999)

1 Maximum number of vehicles: $m = 25$. Time window penalty: $\rho_{TW} = 8.0$; load penalty $\rho_{LD} = 10.0$.
2 Maximum iterations $k = 2500$.
3 Seconds on a Pentium II 400 MHz system. Total runtime ~ 550 seconds each.
4 Seconds on a Sun Sparc 1 workstation.
5 OST is ordered starting tour. RST is random starting tour seeded with the value given. LOT is listed ordering.

UAV Results

We analyzed a Bosnia UAV scenario provided by Bergdahl (1998). Winds for the region of interest are given in Table 7. These winds are taken from actual US Air Force meteorological conditions for the operating region.

Table 7. Wind Data

Altitude Tier	Altitude (ft)	Θ_{WS} (deg)	*WS* (kts)	*AS* (kts)
0	5,000	300	7.5	70
1	10,000	300	37.5	70
2	18,000	310	50.0	70

Scenario details are listed in Table 8, and a map of this scenario is provided in Figure 9. The 52 targets fall into three remote operating zones (ROZs), each with non-overlapping time windows. Route optimization begins and ends with the Srbac, Bosnia waypoint, since the route to and from there must follow a mandatory air corridor.

The scenario was solved in 108 seconds on a Pentium II 300 MHz system using the UAV specific module of the heuristic. With optimum use of wind tiers, the solver returned a tour requiring only one vehicle with a mission time of 822 minutes. Without wind tier modeling, two vehicles are required with a combined mission time of 1384 minutes. This demonstrates the improvement that can be achieved with smart selection of travel altitudes.

The optimized tour output is listed in Table 9 (the "Alt" column designates the altitude tier to be used enroute to the next target); this flight path is shown in Figures Figure 10 and Figure 11. Figure 11 shows the same sequence as Figure 10 with a temporal component as the added third axis and gray bars representing the time windows.

CONCLUSIONS

Our Java implementation of a reactive tabu search first described by Battiti and Tecchiolli (1994) successfully solves single/multiple traveling salesman problems with and without time windows (TSP, MTSP,TSPTW, MTSPTW), as well as capacitated vehicle routing problems with and without time windows (VRP, VRPTW). On the Solomon problem sets, our heuristic produces close to optimal solutions within reasonable computing times. Addition of reactive penalties allows the algorithm to perform more robustly over a wider set of problems.

Our implementation supports UAV problems and formats as well as classical problems and formats. Changes required for the UAV problem reflect unique aspects of the operational UAV mission and include items such as a reformulated objective function, alternate coordinate and numerical formatting, and random customer service times. The introduction of altitude-based wind tiers, when selecting UAV routes, capitalizes on the altitude-dependent, highly asymmetric nature of the flight environment.

The Java implementation is an object oriented structure that is both machine

portable and readily modifiable to support new problem instances. The internal data structure and methodology work with the GUI to support operational requirements such as route locking and dynamic rescheduling in support of priority targets.

We present several ideas that represent natural and worthwhile extensions to the work accomplished.

Heuristic Modifications

Modifications to the tabu search heuristic could include any of the following ideas. Additional operators have increased solution quality for genetic algorithms (Petridis et al. 1998); construction and implementation of additional operators may prove useful. These operators could consider additional random or directed moves which expand the neighborhood, such as a 4-opt, for possible improvements in the objective function.

Restarts based on changes in the solution quality or stabilization of the objective function could prove useful. Methods to consider include the following: maintenance of an *elite list* of best solutions where a restart resumes with relaxed tabu restrictions (Xu and Kelly 1996); intensification from previous location with stored tabu status (Armentano and Ronconi 1999); or a multi-start *backjump tracking* scheme (Liaw 1999, Norwicki and Smutnicki 1996). Other initialization methods such as a sweep initialization or petal initialization (Renaud 1996b) could be explored.

UAV Related Modifications

Changes to the UAV specific aspect of the problem could include a priority scheme hierarchy that generates route segments based on assigned target priorities. This would involve constructing subtours that are then smartly linked together—obviously, the parameters of one subtour will be highly dependent on the others. The rudimentary wind modeling (discrete levels and average regional values) could be replaced with wind values that correlate specifically to each leg. A more detailed modeling of actual UAV transition times between altitudes, with modeled climb rates would provide a higher fidelity mission profile. The service times distribution model, which is still rather unknown, could be updated to reflect data gathered from recent operations.

Java Code Modifications

Although we are not strict computer programmers, work was done in an attempt to improve the code for better heuristic performance. This includes items such as ordering logic comparisons (so that the most likely outcome is encountered first to reduce comparisons) and changing several methods (to decrease instantiations). These optimization modifications reduced the run time for the 100 customer, 25 vehicle Solomon problem sets from an average of 700 seconds to 550 seconds. While this represents about a 25% reduction in run time, there is still tremendous room for improvement due to excessive object copying.

With any Java *non-primitive type*, the statement "x = y" will cause the "x" label to point to the "y" object, and the previous "x" object (if any) will be lost. What remains is the "y" object with both an "x" label and a "y" label. In order to have a separate "x" object that is the same as the "y" object, an explicit copy or clone function must be used to duplicate the object (Flanagan 1997). This duplication is an expensive operation, as it instantiates a new object and copies the member data. Analysis of the current reactive tabu

heuristic with the KL Group's JProbe™ Java profiler tool revealed that nearly 50% of the run time is spent copying NodeType objects. Initial experimentation using an index system as node pointers showed potential run times that are only 20% of the current run time—100 customer, 25 vehicle Solomon sets ran in ~120 seconds versus the current ~550 seconds. This speed increase results from copying and manipulating the indices, which are Java primitive types, instead of copying and manipulating the NodeType objects.

Some initial work was done in an attempt to reconfigure the heuristic to run using indices, but the changes are substantial as they touch nearly every aspect of the program. Future programming efforts should make the conversion, which will allow faster solutions and larger problem sets.

Table 8. Bosnia Data Set

Location Name	Lat	(DD	MM	SS)	Lon	(DD	MM	SS)	e_i[1]	ℓ_i[1]	$s_{min}(i)$[2]	$s_{max}(i)$[2]
DepotTazarHungary	N	46	24	0	E	17	54	0				
CorridorSzulokHungary	N	46	3	45	E	17	32	44				
CorridorSrbacBosnia*	N	45	24	0	E	17	30	0	940	4800	0	0
Dumdvga	N	44	58	29	E	16	50	34	1015	1500	30	180
Mastye	N	44	58	46	E	16	38	56	1015	1500	30	180
AAASiteGarred	N	44	58	4	E	16	39	31	1015	1500	2	15
HvyWpnDepTharmet	N	44	58	33	E	16	39	18	1015	1500	2	30
HvyWpnDepTharmet	N	44	58	39	E	16	39	41	1015	1500	2	30
HvyWpnDepTharmet	N	44	58	59	E	16	39	28	1015	1500	2	30
CommSiteSardona	N	44	59	2	E	16	39	56	1015	1500	2	30
CommSiteSardona	N	44	59	11	E	16	40	19	1015	1500	2	30
CommSiteSardona	N	44	59	15	E	16	39	20	1015	1500	2	30
SuspWpnStorage	N	44	59	9	E	16	39	10	1015	1500	2	30
SuspWpnStorage	N	44	54	52	E	16	34	47	1015	1500	2	30
SuspWpnStorage	N	44	51	49	E	16	41	37	1015	1500	2	30
SuspWpnStorage	N	45	0	7	E	16	34	47	1015	1500	2	30
SuspWpnStorage	N	44	59	9	E	16	49	17	1015	1500	2	30
SuspWpnStorage	N	44	57	41	E	16	39	35	1015	1500	2	30
SAMIADSiteProbSA2	N	44	57	23	E	16	51	45	1015	1500	2	30
SAMIADSiteProbSA2	N	44	57	45	E	16	49	28	1015	1500	2	30
SAMIADSiteProbSA2	N	44	55	57	E	16	43	52	1015	1500	2	30
SAMIADSiteSiteRadar	N	44	57	47	E	16	39	54	1015	1500	2	30
HQSiteDromada	N	45	0	7	E	16	53	49	1015	1500	30	120
WarehouseDromada	N	44	53	31	E	16	54	12	1015	1500	2	60
BarracksOmanski	N	44	45	34	E	17	10	34	1500	1715	5	120
BarracksOmanski	N	44	48	19	E	17	12	14	1500	1715	5	120
BarracksOmanski	N	44	51	2	E	17	13	24	1500	1715	5	120
TankRallyPointBolstavec	N	44	50	51	E	17	14	39	1500	1715	2	30
TankRallyPointBolstavec	N	44	56	17	E	17	17	41	1500	1715	2	30
StorageBunkerKrajachastane	N	44	55	51	E	17	17	51	1500	1715	2	30
StorageBunkerKrajachastane	N	44	56	7	E	17	18	23	1500	1715	2	30
RoadGolprtuniy	N	44	28	13	E	17	1	18	1730	1830	20	40
RoadGolprtuniy	N	44	27	29	E	17	1	46	1730	1830	20	40
RoadGolprtuniy	N	44	27	10	E	17	2	24	1730	1830	20	40
Dumdvga	N	44	58	29	E	16	50	34	1900	2300	30	180
Mastye	N	44	58	46	E	16	38	56	1900	2300	30	180
AAASiteGarred	N	44	58	4	E	16	39	31	1900	2300	2	15
HvyWpnDepTharmet	N	44	58	33	E	16	39	18	1900	2300	2	30
HvyWpnDepTharmet	N	44	58	39	E	16	39	41	1900	2300	2	30
HvyWpnDepTharmet	N	44	58	59	E	16	39	28	1900	2300	2	30
CommSiteSardona	N	44	59	2	E	16	39	56	1900	2300	2	30
CommSiteSardona	N	44	59	11	E	16	40	19	1900	2300	2	30
CommSiteSardona	N	44	59	15	E	16	39	20	1900	2300	2	30
SuspWpnStorage	N	44	59	9	E	16	39	10	1900	2300	2	30
SuspWpnStorage	N	44	54	52	E	16	34	47	1900	2300	2	30
SuspWpnStorage	N	44	51	49	E	16	41	37	1900	2300	2	30
SuspWpnStorage	N	45	0	7	E	16	34	47	1900	2300	2	30
SuspWpnStorage	N	44	59	9	E	16	49	17	1900	2300	2	30
SuspWpnStorage	N	44	57	41	E	16	39	35	1900	2300	2	30
SAMIADSiteProbSA2	N	44	57	23	E	16	51	45	1900	2300	2	30
SAMIADSiteProbSA2	N	44	57	45	E	16	49	28	1900	2300	2	30
SAMIADSiteProbSA2	N	44	55	57	E	16	43	52	1900	2300	2	30
SAMIADSiteSiteRadar	N	44	57	47	E	16	39	54	1900	2300	2	30
HQSiteDromada	N	45	0	7	E	16	53	49	1900	2300	30	120
WarehouseDromada	N	44	53	31	E	16	54	12	1900	2300	2	60
CorridorSrbacBosnia	N	45	24	0	E	17	30	0	940	4740	0	0
DepotTazarHungary	N	46	24	0	E	17	54	0				
CorridorSzulokHungary	N	46	3	45	E	17	32	44				

(Bergdahl 1998)

[1] Time listed in *hours-minutes* format.

[2] Minutes.

* Optimization begins from Srbac Corridor waypoint

Figure 9. Bosnia Scenario Target Locations

Table 9. Bosnia Tour Sequence

Label	ID	Lat[2]	Long[3]	Early[4]	Late[4]	Arr[4]	Dep[4]	Serv[4]	Alt[5]
CorridorSrbacBosnia	0	45.1166	-17.5416	580	2880	580.00	580.00	0	0
HQSiteDromada	20	45.0019	-16.8969	615	900	606.25	615.00	30	0
SAMIADSiteProbSA2	16	44.9563	-16.8625	615	900	647.66	647.66	2	0
Dumdvga	1	44.9747	-16.8427	615	900	650.97	650.97	125	0
SuspWpnStorage	14	44.9858	-16.8213	615	900	778.02	778.02	2	2
SAMIADSiteProbSA2	17	44.9625	-16.8244	615	900	780.89	780.89	2	0
SAMIADSiteProbSA2	18	44.9325	-16.7311	615	900	786.88	786.88	2	0
CommSiteSardona	8	44.9863	-16.6719	615	900	792.77	792.77	2	0
CommSiteSardona	7	44.9838	-16.6655	615	900	795.05	795.05	2	0
CommSiteSardona	9	44.9875	-16.6555	615	900	797.50	797.50	6	0
SuspWpnStorage	10	44.9858	-16.6527	615	900	804.11	804.11	3	2
HvyWpnDepTharmet	6	44.983	-16.6577	615	900	807.86	807.86	2	2
HvyWpnDepTharmet	5	44.9775	-16.6613	615	900	810.05	810.05	2	2
SAMIADSiteSiteRadar	19	44.963	-16.665	615	900	812.57	812.57	2	0
SuspWpnStorage	15	44.9613	-16.6597	615	900	814.79	814.79	2	0
AAASiteGarred	3	44.9677	-16.6586	615	900	817.14	817.14	2	0
HvyWpnDepTharmet	4	44.9758	-16.6549	615	900	819.61	819.61	2	0
Mastye	2	44.9794	-16.6488	615	900	821.93	821.93	30	0
SuspWpnStorage	13	45.0019	-16.5797	615	900	855.02	855.02	19	2
SuspWpnStorage	11	44.9144	-16.5797	615	900	878.36	878.36	2	2
SuspWpnStorage	12	44.8636	-16.6936	615	900	883.24	883.24	2	1
WarehouseDromada	21	44.8919	-16.9033	615	900	891.03	891.03	2	1
TankRallyPointBolstavec	26	44.938	-17.2947	900	1035	903.71	903.71	2	2
StorageBunkerKrajachastane	28	44.9352	-17.3063	900	1035	905.98	905.98	15	0
StorageBunkerKrajachastane	27	44.9308	-17.2975	900	1035	921.88	921.88	2	0
TankRallyPointBolstavec	25	44.8475	-17.2441	900	1035	928.56	928.56	2	0
BarracksOmanski	24	44.8505	-17.2233	900	1035	931.42	931.42	15	2
BarracksOmanski	23	44.8052	-17.2038	900	1035	949.23	949.23	5	0
BarracksOmanski	22	44.7594	-17.1761	900	1035	956.77	956.77	5	2
RoadGolprtuniy	31	44.4527	-17.04	1050	1110	977.93	1050.00	20	0
RoadGolprtuniy	30	44.458	-17.0294	1050	1110	1070.52	1070.52	20	0
RoadGolprtuniy	29	44.4702	-17.0216	1050	1110	1091.27	1091.27	22	0
SuspWpnStorage	43	44.8636	-16.6936	1140	1380	1139.59	1140.00	13	0
SuspWpnStorage	42	44.9144	-16.5797	1140	1380	1159.13	1159.13	2	0
SuspWpnStorage	44	45.0019	-16.5797	1140	1380	1165.90	1165.90	2	2
Mastye	33	44.9794	-16.6488	1140	1380	1169.55	1169.55	30	0
SuspWpnStorage	41	44.9858	-16.6527	1140	1380	1199.91	1199.91	24	0
CommSiteSardona	40	44.9875	-16.6555	1140	1380	1224.17	1224.17	2	2
CommSiteSardona	39	44.9863	-16.6719	1140	1380	1226.56	1226.56	2	0
CommSiteSardona	38	44.9838	-16.6655	1140	1380	1228.84	1228.84	21	0
HvyWpnDepTharmet	37	44.983	-16.6577	1140	1380	1250.84	1250.84	2	2
HvyWpnDepTharmet	36	44.9775	-16.6613	1140	1380	1253.03	1253.03	8	0
HvyWpnDepTharmet	35	44.9758	-16.6549	1140	1380	1262.16	1262.16	2	2
AAASiteGarred	34	44.9677	-16.6586	1140	1380	1264.44	1264.44	2	2
SuspWpnStorage	46	44.9613	-16.6597	1140	1380	1266.67	1266.67	2	1
SAMIADSiteSiteRadar	50	44.963	-16.665	1140	1380	1268.84	1268.84	2	2
SAMIADSiteProbSA2	49	44.9325	-16.7311	1140	1380	1272.52	1272.52	2	2
WarehouseDromada	52	44.8919	-16.9033	1140	1380	1278.57	1278.57	2	0
SAMIADSiteProbSA2	47	44.9563	-16.8625	1140	1380	1284.55	1284.55	2	0
SAMIADSiteProbSA2	48	44.9625	-16.8244	1140	1380	1288.13	1288.13	28	0
SuspWpnStorage	45	44.9858	-16.8213	1140	1380	1318.38	1318.38	2	2
Dumdvga	32	44.9747	-16.8427	1140	1380	1320.94	1320.94	30	1
HQSiteDromada	51	45.0019	-16.8969	1140	1380	1353.14	1353.14	30	0
CorridorSrbacBosnia		45.1166	-17.5416	580	2880	1401.57	—	0	—

[1] Parameters set as follows: maximum number of vehicles: $m = 5$; maximum iterations: $k = 2500$; reactive penalty scheme; LOT starting tour. Total runtime 108 seconds on a Pentium II 300 MHz system.
[2] By convention North latitudes are positive and South latitudes are negative.
[3] By convention, West longitudes are positive and East longitudes are negative.
[4] Time in minutes.
[5] Flight altitude to next point: "0" = 5,000 ft, "1" = 10,000 ft, "2" = 18,000 ft.

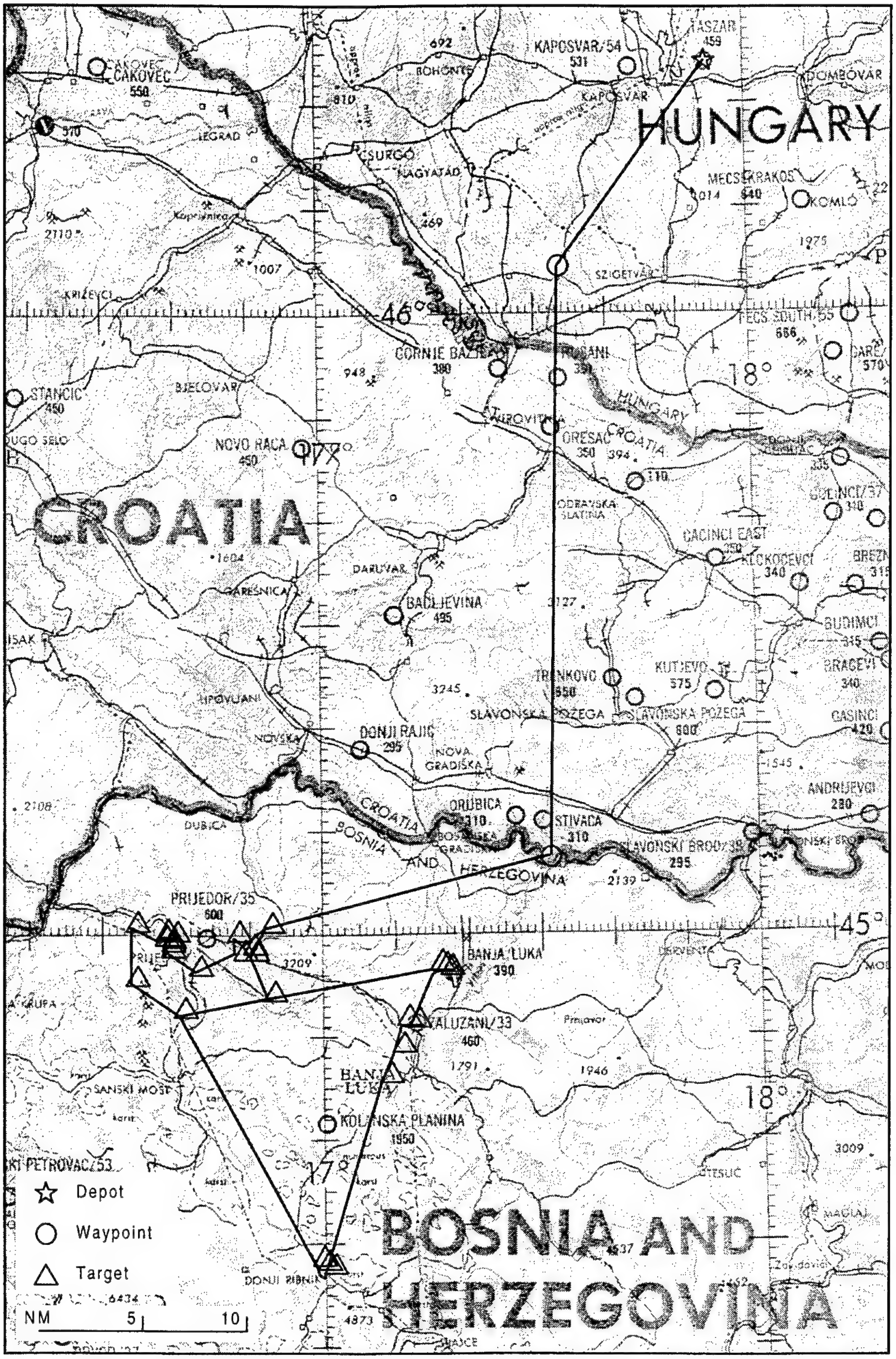


Figure 10. Bosnia Optimized Tour Route

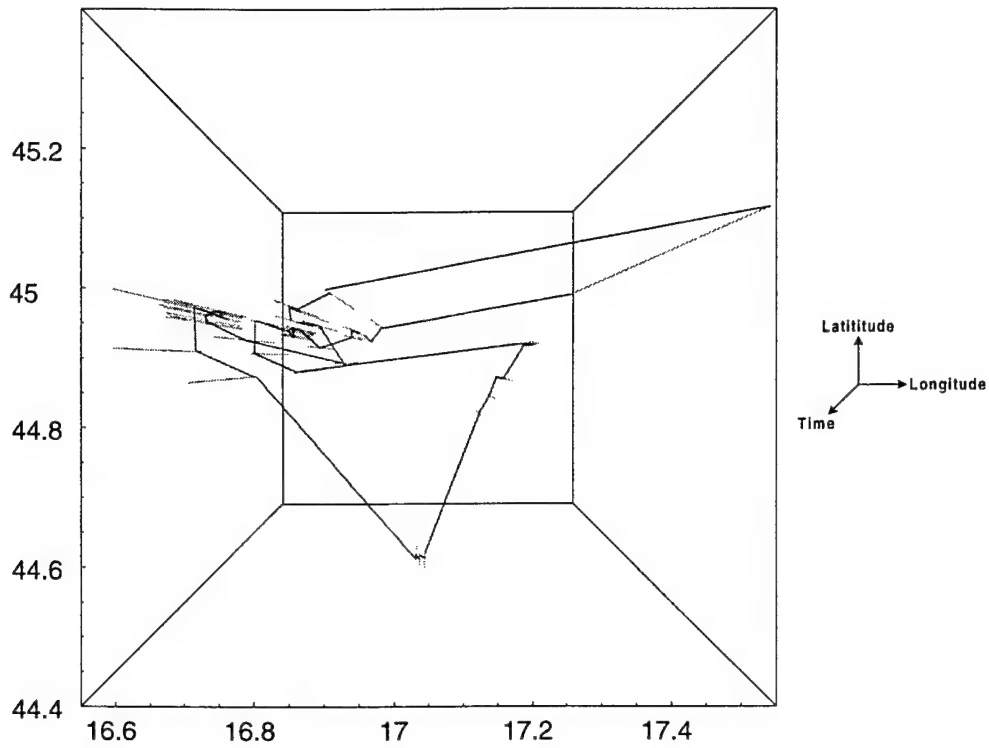


Figure 11. Temporal Route Plot

BIBLIOGRAPHY

Armenato, Vinícius A., and Débora P. Ronconi. "Tabu Search for Total Tardiness Minimization in Flowshop Scheduling Problems," *Computers & Operations Research, 26*: 219-235 (1999).

Barbarosoglu, G. and D. Ozgur. "A tabu search algorithm for the vehicle routing problem," *Computers & Operations Research, 26*: 255-270 (1999).

Battiti, R. R., and G. Tecchiolli. "The Reactive Tabu Search," *ORSA Journal on Computing, 6*: 126-140 (1994).

Bergdahl, B. Operations Officer, 11th Reconnaissance Squadron, Indian Springs AFAF NV. Personal interview and facsimile. January 1998.

Bodin, Lawrence, Bruce Golden, A. Assad, and M. Ball. "Routing and Scheduling of Vehicles and Crews; The State of the Art," *Computers & Operations Research, 10*: (1983).

Carlton, William B. *A Tabu Search to the General Vehicle Routing Problem.* Ph.D. dissertation. University of Texas, Austin TX, 1995.

Chiang, W. C. and R. Russell. "A Reactive Tabu Search Metaheuristic for the Vehicle Routing Problem with Time Windows," *ORSA Journal on Computing, 9*: 417 (1997).

Desrochers, M., J. Desrosiers, and M. Solomon. "A New Optimization Algorithm for the Vehicle Routing Problem with Time Windows," *Operations Research, 40*: 342-354 (1992).

Departments of the Air Force and Navy. Flying Training, Air Navigation. Air Force Regulation (AFR) 51-40. Washington: HQ USAF, 15 Mar 1983.

Eckel, Bruce. *Thinking in Java—The definitive introduction to object-oriented programming in the language of the World-Wide Web.* Upper Saddle River NJ: Prentice-Hall, 1998.

Flanagan, David. *Java in a Nutshell, A Desktop Quick Reference* (Second Edition). Sebastopol CA: O'Reilly & Associates, 1997.

Garcia, B. L., J. Y. Potvin, and J. M. Rousseau. "A Parallel Implementation of the Tabu Search Heuristic for Vehicle Routing Problems with Time Window Constraints," *Computers & Operations Research, 21*: 1025-1033 (1994).

Gendreau, M., A. Hertz, and G. Laporte. "Tabu Search Heuristic for the Vehicle Routing Problem," *Management Science, 40*: 1276-1289 (October 1994).

Gendreau, M., G. Laporte, and R. Séguin. "A Tabu Search Heuristic for the Vehicle Routing Problem with Stochastic Demands and Customers," *Operations Research, 44*: 469-477 (May 1996).

Gendreau, M., G. Laporte, and G. Potvin. "Vehicle Routing: Modern Heuristics," in *Local Search in Combinatorial Optimization.* Eds. Aarts, E. and J. K. Lenstra. Chichester: Wiley, 1997.

Glover, Fred and M. Laguna. *Tabu Search.* Boston: Kluwer Academic Publishers, 1997.

Glover, Fred. "Tabu Search-Part I," *ORSA Journal on Computing, 1*: 190-206 (Summer 1989).

Liaw, C. "A Tabu Search Algorithm for the Open Shop Scheduling Problem," *Computers & Operations Research, 26*: 109-126 (1999).

McKenna, P. "Eyes of the Warrior—Prying Predator prowls unfriendly skies, peeking at the enemy," *Airman, XLII(7)*: 28-31 (July 1998).

Norwicki, E. and C. Smutnicki. "A Fast Taboo Search Algorithm for the Job Shop Problem," *Management Science, 42*: 797-813 (1996).

Osman, I. H. "Metastrategy Simulated Annealing and Tabu Search Algorithms for the Vehicle Routing Problem," *Annals of Operations Research, 41*: 421-451 (1993).

Parsons, T. L. Meteorologist, US Air Force, Air Force Institute of Technology, Wright Patterson AFB OH. Personal interview. 23 February 1999.

Petridis, V., S. Kazarlis, and A. Bakirtzis. "Varying Fitness Functions in Genetic Algorithm Constrained Optimization: The Cutting Stock and Unit Commitment Problems," *IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics, 28(5)*: 629-640 (October 1998).

Renaud, J., G. Laporte, and F. Boctor. "A Tabu Search Heuristic for the Multi-Depot Vehicle Routing Problem," *Computers & Operations Research, 23*: 229-235 (1996a).

Renaud, J., F. Boctor, and G. Laporte. "An Improved Petal Heuristic for the Vehicle Routeing [sic] Problem," *Journal of the Operations Research Society, 47*: 329-336 (1996b).

Rochat, Y. and F. Semet. "A Tabu Search Approach for Delivering Pet Food and Flour in Switzerland," *Journal of Operations Research Society, 45*: 1233-1246 (1994).

Ryan, Joel L. *Embedding a Reactive Tabu Search Heuristic in Unmanned Aerial Vehicle Simulations*. MS thesis, AFIT/GOR/ENS/98M. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, February 1998.

Ryer, David M. *Implementation of the Metaheuristic Tabu Search in Route Selection for Mobility Analysis Support System.* MS thesis, AFIT/GOA/ENS/99M-07. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 1999.

Sisson, M. R. *Applying Tabu Heuristic to Wind Influenced, Minimum Risk and Maximum Expected Coverage Routes.* MS thesis, AFIT/GOR/ENS/97M. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, February 1997.

Xu, J. and J. Kelly. "A Network Flow-Based Tabu Search Heuristic for the Vehicle Routing Problem," *Transportation Science, 30*: 379-393 (November 1996).

Woodruff, D. and E. Zemel. "Hashing Vectors for Tabu Search," *Annals of Operations Research, Vol. 41*: 123-137 (1993).

Notation

The following symbols appear in the main body of the paper and are defined as listed. For the sake of clarity, symbols appearing only in the appendices are defined when introduced.

a_i	= arrival time at node i
A	= arc set
A	= line segment, wind adjustment triangle
AS	= air speed
B	= line segment, wind adjustment triangle
c_{ij}	= cost (travel time) from node i to j
C	= line segment, wind adjustment triangle
C_i	= excess vehicle capacity
d	= insertion move depth
d_i	= departure time from node i
d_{ij}	= great circle distance between locations i and j
D	= maximum tour-length duration
D_i	= excess route duration
e_i	= earliest *begin service* time of node i
G	= graph set
GS	= ground speed
H	= intermediate heading angle
h_{ijk}	= travel altitude k between locations i and j
i	= insertion move position
k	= altitude band
k	= iteration
ℓ_i	= latest *begin service* time of node i
L	= latitude
LD	= load overage violations
m	= number of vehicles
n	= number of customers
q_i	= non-negative customer demand (quantity) of node i
Q	= vehicle capacity
s_i	= customer service time of node i
$s_{max}(i)$	= maximum stochastic service time for location i
$s_{min}(i)$	= minimum stochastic service time for location i
S_i	= stochastic customer service time for location i
t	= arbitrary time within time window
t_{ij}	= travel time from node i to j
t_{LD}	= number of load infeasible solutions in the previous ten iterations
t_{TW}	= number of time window infeasible solutions in the previous ten iterations
TW	= time window violations
$thv(t)$	= tour hashing value
v_0	= initial depot node (TSP, VRP)

v_i	=	additional nodes (TSP, VRP)
V	=	vertex set
w_i	=	wait time to commence at node i
WS	=	wind speed
x_{ij}	=	indicator variable denoting arc from node i to j is included in the tour
$Z_f(t)$	=	feasible objective function
$Z(t)$	=	penalized objective function
$Z'_f(t)$	=	feasible UAV objective function
$Z'(t)$	=	penalized UAV objective function
Θ_{ij}	=	bearing from location i to j
Θ_{WS}	=	wind bearing
Ψ_{ij}	=	random weight for arc i, j
δ	=	course correction angle
λ	=	longitude
θ	=	tabu list length
ρ	=	generic penalty scaling factor
ρ_{LD}	=	load capacity penalty scaling factor
ρ_{TW}	=	time window penalty scaling factor
τ	=	tour position

Appendix A. Extended Problem Formulation

This appendix examines the formulation of the traveling salesman problem (TSP), multiple traveling salesman problem (MTSP), vehicle routing problem (VRP), and multiple-depot vehicle routing problem (MDVRP). It is provided for generality and thoroughness as these problem types have additional constraints which were not mentioned previously since they are intrinsicly modeled in the tabu search heuristic. For instance, based on the way the tabu search evaluates the neighborhood and swaps customers, no subtour breaking constraint is provided since it is impossible for the heuristic to construct a subtour. Notation and numbering of variables differs slightly from that presented in Chapter 2, as the notation there is tailored to the problem context.

A.1 Traveling Salesman Problem (TSP)

The first problem class, and basis for the remaining types, is the traveling salesman problem (TSP). Begin by defining the TSP structure and objective as follows: Let G be our network with the set of nodes N, a set of branches A, and the associated non-negative branch costs of $C = c_{ij}$. The objective of this problem is to form a tour spanning all the nodes beginning and ending at the origin (node 1), which yields the minimum total tour length or cost. In the most basic case, we assume that the costs are symmetric ($c_{ij} = c_{ji}$), but the problem can be asymmetric with no loss of generality.
This can be represented as an assignment problem, where exactly one arc x_{ij} starts at node i, and exactly one arc x_{ij} terminates at node j. Specifically, the problem is formulated as follows:

$$\text{Minimize } Z(t) = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \tag{A1.1}$$

Where

$$x_{ij} = \begin{cases} 1 & \text{if arc } ij \text{ is in the tour} \\ 0 & \text{otherwise} \end{cases}$$

Subject to:

$$\sum_{i=1}^{n} x_{ij} = b_j = 1 \quad (j = 1,2,\ldots,n) \tag{A1.2}$$

$$\sum_{j=1}^{n} x_{ij} = a_j = 1 \quad (j = 1,2,\ldots,n) \tag{A1.3}$$

Where

$$X = (x_{ij}) \in S, \qquad x_{ij} \in \{0,1\} \qquad \forall\ i,j = 1,2,\ldots,n \ .$$

As previously mentioned, additional constraints are required to eliminate subtours. Adding the subtour breaking constraint to the assignment formulation prevents

subtours. The three standard ways to represent a subtour breaking constraint (Bodin et al. 1983) are listed as follows:

(1) $S = \left\{ (x_{ij}) : \sum_{i \in Q} \sum_{j \notin Q} x_{ij} \geq 1 \text{ for every nonempty proper subset } Q \text{ of } N \right\}$

(2) $S = \left\{ (x_{ij}) : \sum_{i \in R} \sum_{j \in R} x_{ij} \leq |R| - 1 \text{ for every nonempty subset } R \text{ of } \{2,3,\ldots,n\} \right\}$

(3) $S = \{(x_{ij}) : y_i - y_j + n x_{ij} \leq n - 1 \text{ for } 2 \leq i \neq j \leq n \text{ for some real numbers } y_i\}$.

The first constraint requires that every node subset Q of the solution set be connected to all of the other nodes in the solution. The second constraint requires that the arcs in the solution set contain no cycles (a cycle over R nodes must contain $|R|$ arcs. The third constraint is not intuitively straightforward and calls for more explanation. First, define y_i as follows:

$$y_i = \begin{cases} t & \text{if node } i \text{ is visited on the } t^{\text{th}} \text{ step in a tour} \\ 0 & \text{otherwise} \end{cases}$$

For an arc in the solution tour ($x_{ij} = 1$), the constraint becomes

$$t - (t+1) + n \leq n - 1 \ .$$

For an arc not contained in the solution tour ($x_{ij} = 0$), the constraint reduces to

$$y_i - y_j \leq n - 1 \ .$$

The third representation has the advantage of adding only $n^2 - 3n + 2$ subtour breaking constraints to the formulation, where the previous two add 2^n constraints (Bodin et al. 1983).

A.2 Multiple Traveling Salesman Problem (MTSP)

Adding more salesmen to the problem gives the next level of complexity, the multiple traveling salesman problem (MTSP). Let m be the number of salesmen or vehicles that make up the fleet. Again the objective is to minimize the total distance traveled. Assume further that the m salesmen depart from and return to the same depot and that each customer must be visited exactly once by exactly one salesman.

With these changes, the formulation is an extension of the basic TSP presented above and is represented as

$$\text{Minimize } Z(t) = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \qquad \text{(A2.1)}$$

Subject to:

$$\sum_{i=1}^{n} x_{ij} = b_j = \begin{cases} M & \text{if } j = 1 \\ 1 & \text{if } j = 2,3,\ldots,n \end{cases} \qquad \text{(A2.2)}$$

$$\sum_{j=1}^{n} x_{ij} = a_j = \begin{cases} M & \text{if } j = 1 \\ 1 & \text{if } j = 2,3,\dots,n \end{cases} \quad \text{(A2.3)}$$

where $X = (x_{ij}) \in S$, $x_{ij} \in \{0,1\}$ $\forall\ i, j = 1,2,\dots,n$.

The first constraint in the formulation requires that all salesmen be used by forcing them to leave the depot. The second constraint requires all salesmen to return to the depot. Any one of the subtour breaking constraints used earlier in the TSP can be used for the MTSP.

The apparent complexity of this new problem can be reduced by representing the MTSP as m copies of the single TSP. This is accomplished by creating dummy depots ($D_1,\dots,D_m$) that are connected to the original network. These m copies are either separate from each other, or are connected with cost prohibitive *big M* arcs. When these single TSP copies are connected to a common depot, the problem becomes a series of m subtours, which when taken together forms the MTSP. This relatively straightforward transformation of the MTSP helps demonstrates why a TSP algorithm can be used to solve MTSP problems (Bodin et al. 1983).

A.3 Vehicle Routing Problem (VRP)

The next extension of the TSP is the Vehicle Routing Problem (VRP) which is obtained by adding a capacity constraint to the salesman or vehicles. In the VRP, a number of vehicles w leave a depot and service a number of customers n, each possessing a unique demand d_i. Each vehicle v has a limited capacity K_v and a maximum route duration T_v that constrains their closed delivery routes, or return to depot time. This particular instance of the VRP is commonly known as the *general vehicle routing problem* (GVRP). If the maximum route lengths or range constraints are removed, then this problem is referred to as the *standard vehicle routing problem* (SVRP) (Bodin et al. 1983). Additionally, the time required for a vehicle v to deliver or service at node i is s_i^v, the travel time for vehicle v from node i to node j is t_{ij}^v, and finally $x_{ij}^v = 1$ if arc i-j is used by vehicle v. From this, the formulation of the GVRP is as follows:

$$\text{Minimize } Z(t) = \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{v=1}^{w} c_{ij} x_{ij}^{v} \quad \text{(A3.1)}$$

Subject to:

$$\sum_{i=1}^{n}\sum_{v=1}^{w} x_{ij}^{v} = 1 \quad (j = 2, \dots, n) \quad \text{(A3.2)}$$

$$\sum_{j=1}^{n}\sum_{v=1}^{w} x_{ij}^{v} = 1 \quad (i = 2, \dots, n) \quad \text{(A3.3)}$$

$$\sum_{i=1}^{n} x_{ip}^{v} - \sum_{j=1}^{n} x_{pj}^{v} = 0 \quad (v = 1, \dots, w;\ p = 1, \dots, n) \quad \text{(A3.4)}$$

$$\sum_{i=1}^{n} d_i \left(\sum_{j=1}^{n} x_{ij}^{v}\right) \le K_v \quad (v = 1, \dots, w) \quad \text{(A3.5)}$$

$$\sum_{i=1}^{n} s_i^v \sum_{j=1}^{n} x_{ij}^v + \sum_{i=1}^{n}\sum_{j=1}^{n} t_{ij}^v x_{ij}^v \le T_v \quad (v = 1, \dots, w) \tag{A3.6}$$

$$\sum_{j=2}^{n} x_{1j}^v \le 1 \quad (v = 1, \dots, w) \tag{A3.7}$$

$$\sum_{i=2}^{n} x_{i1}^v \le 1 \quad (v = 1, \dots, w) \tag{A3.8}$$

where $X = \left(x_{ij}^v\right) \in S$, $x_{ij}^v \in \{0,1\}$ $\forall\ i, j, v$.

The objective function, which minimizes the overall distance, remains the same but is formulated to sum over all vehicles. Equations (A3.2) and (A3.3) require that every customer is visited by exactly one vehicle. It is assumed that a customer's demand does not exceed vehicle capacity and that each customer is fully serviced by the single vehicle that visits it. Equation (A3.4) requires continuity of our routes while (A3.5) maintains the vehicle capacity constraint. Since route length restrictions are represented with times, equation (A3.6) requires that maximum route duration is not exceeded. Finally, equations (A3.7) and (A3.8) limit the number of vehicles used.

In addition to these equations, subtour breaking constraints, slightly modified from those used earlier in the TSP, must be included. Since it is the most efficient, the third subtour representation is selected for expansion as follows:

$$S = \left\{x_{ij}^v : y_i^v - y_j^v + n x_{ij}^v \le n-1 \text{ for } 2 \le \mathrm{i} \ne \mathrm{j} \le \mathrm{n} \text{ for some real numbers } y_i^v \right\}$$

This applies the original subtour breaking constraint to each vehicle in turn. We note that some redundant constraints can be eliminated from the formulation above. Using (A3.2) and (A3.4) enforces (A3.3) automatically and makes it unnecessary (Bodin et al. 1983). Likewise (A3.4) and (A3.7) imply (A3.8) so this too can be eliminated from the formulation (Bodin et al. 1983).

Finally, one common constraint added to the VRP is time windows. Let a_j be the arrival time to node j, e_j be the earliest delivery time allowable and l_j be the no later than time for delivery. A nonlinear representation yields

$$a_j = \sum_v \sum_i (a_i + s_i^v + t_{ij}^v) x_{ij}^v \quad (j = 1, 2, \dots, n) \tag{A3.9}$$

$$a_1 = 0 \tag{A3.10}$$

$$e_j \le a_j \le l_j \quad (j = 2, \dots, n) \ . \tag{A3.11}$$

If $x_{ij}^v = 0$ then $a_j = 0$. Otherwise a_j is the sum of the previous arrival time ($a_i = 0$), the service time at node i (s_i^v), and the travel time from i to j (t_{ij}^v). Alternatively the linear representation of time windows constraint (Bodin et al. 1983) can be used in the formulation

$$\left.\begin{aligned} a_j &\ge \left(a_i + s_i^v + t_{ij}^v\right) - \left(1 - x_{ij}^v\right) \cdot T_{max}^v \\ a_j &\le \left(a_i + s_i^v + t_{ij}^v\right) + \left(1 - x_{ij}^v\right) \cdot T_{max}^v \end{aligned}\right\} \text{for all } i, j, v \ . \tag{A3.12}$$

When $x_{ij}^{v} = 1$, the second half of the equation is eliminated and a_j is determined from the previous arrival time, previous service time, and the travel time between the nodes. When $x_{ij}^{v} = 0$, the constraints are redundant.

A.4 Multiple Depot Vehicle Routing Problem (MDVRP)

Expanding the previous GVRP to account for multiple depots, or bases of operation, gives the multiple depot VRP. This problem can be formulated with only minor changes. Let M be the number of depots in our problem. First the original VRP formulation indexes are changed for equation (A3.2), ($j = M + 1, \dots, n$), and equation (A3.3), ($i = M + 1, \dots, n$). Next the constraints (A3.7) and (A3.8) are changed to sum over all the depots individually to require that the number of vehicles used does not exceed the number of vehicles available.

$$\sum_{i=1}^{M} \sum_{j=M+1}^{n} x_{ij}^{v} \le 1 \qquad (v = 1, \dots, w)$$

$$\sum_{p=1}^{M} \sum_{i=M+1}^{n} x_{ip}^{v} \le 1 \qquad (v = 1, \dots, w)$$

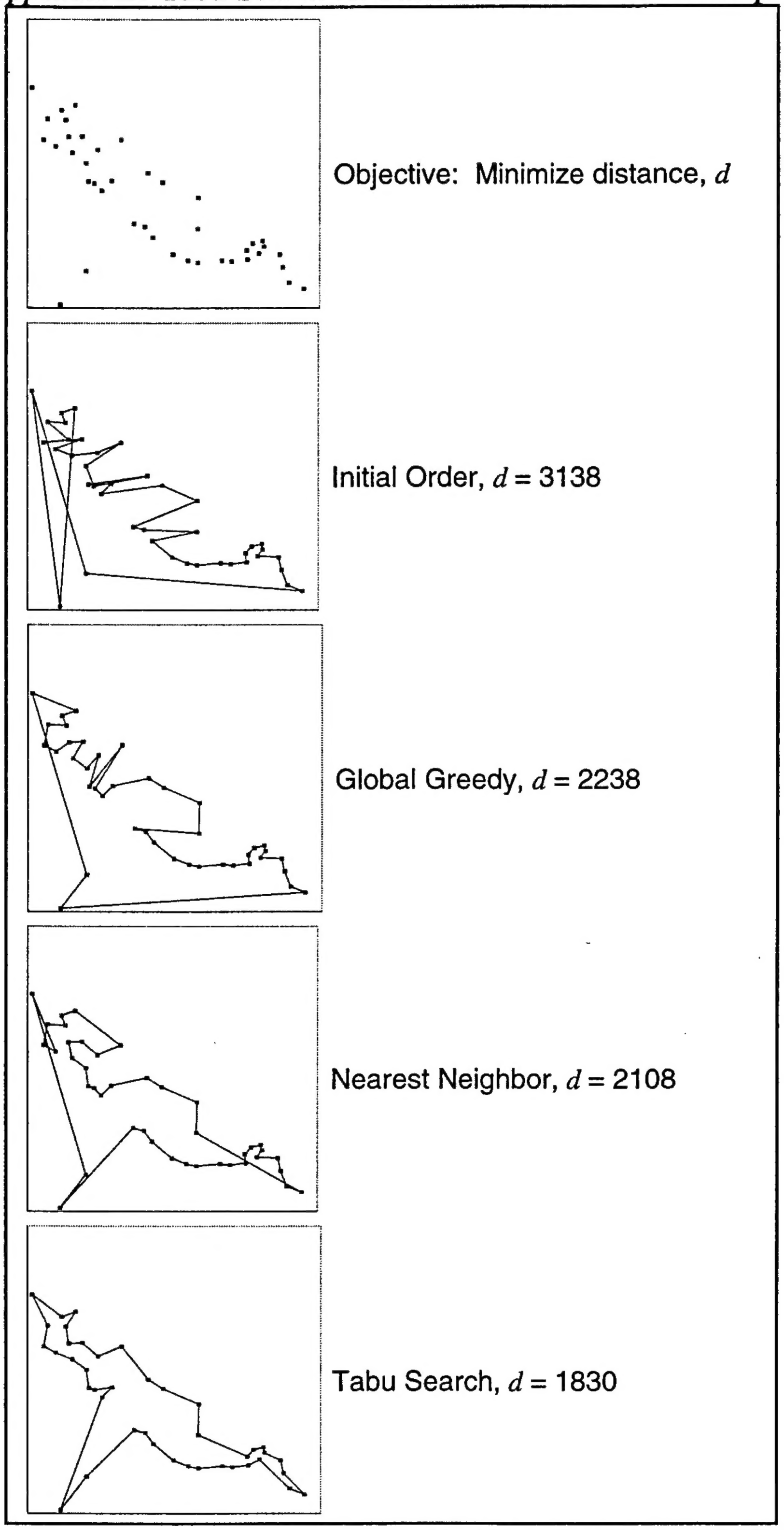
The MDVRP also requires an adjustment to the subtour breaking constraint. Again, only one is required (Bodin et al. 1983).

(1) $S = \{(x_{ij})$: $\sum_{i \in Q} \sum_{j \notin Q} x_{ij} \ge 1$ for every nonempty proper subset Q of $\{1, 2, \dots, n\}$ containing nodes $1, 2, \dots, M\}$;

(2) $S = \{(x_{ij})$: $\sum_{i \in R} \sum_{j \in R} x_{ij} \le |R| - 1$ for every nonempty subset R of $\{M+1, M+2, \dots, n\}\}$;

(3) $S = \{(x_{ij})$: $y_i - y_j + n x_{ij} \le n - 1$ for $M + 1 \le i \ne j \le n$ for some real numbers $y_i\}$.

Appendix B. Tabu Search vs. Other Heuristics—TSP Example



Nari Data Set (Sisson 1997)